高等学校计算机基础教育教材精选

微型计算机原理与接口技术（第4版）

吴宁 乔亚男 主编
冯博琴 主审

清华大学出版社

高等学校计算机基础教育教材精选

微型计算机原理与接口技术
（第4版）

吴　宁　乔亚男　主编
冯博琴　主审

清华大学出版社
北京

内 容 简 介

本书是《微型计算机原理与接口技术》的第4版。本次修订，除对原稿部分文字和有关计算机硬件的新技术进行修订和补充外，首次引入了案例描述模式，将一个实际的系统设计案例贯穿到书中，使全书成为一个有机整体。考虑到读者对象的需求和实用性，本版仍以Intel 80x86系列微处理器为平台，介绍了其三个不同时期的典型代表——8088、80386及Pentium 4的基本结构和工作原理；保持了第3版中的基本指令系统、输入输出系统、接口电路设计的内容以及叙述风格。另外，此次改版，依然保持了本书注重实际应用的特点，实用性较强。

本书可作为普通高等院校理工类各专业本科学生的“微机原理与接口技术”课程的教材，也可作为成人高等教育的培训教材及广大科技工作者的自学参考书。

图书在版编目（CIP）数据

微型计算机原理与接口技术/吴宁，乔亚男主编．--4版．--北京：清华大学出版社，2016(2018.1重印)
高等学校计算机基础教育教材精选
ISBN 978-7-302-44645-3

Ⅰ．①微… Ⅱ．①吴… ②乔… Ⅲ．①微型计算机－理论－高等学校－教材 ②微型计算机－接口技术－高等学校－教材 Ⅳ．①TP36

中国版本图书馆CIP数据核字(2016)第179438号

责任编辑：焦 虹
封面设计：傅瑞学
责任校对：焦丽丽
责任印制：杨 艳

出版发行：清华大学出版社
网 址：http://www.tup.com.cn，http://www.wqbook.com
地 址：北京清华大学学研大厦A座 **邮 编**：100084
社 总 机：010-62770175 **邮 购**：010-62786544
投稿与读者服务：010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈：010-62772015，zhiliang@tup.tsinghua.edu.cn
课件下载：http://www.tup.com.cn,010-62795954
印 装 者：三河市少明印务有限公司
经 销：全国新华书店
开 本：185mm×260mm **印 张**：24.25 **字 数**：556千字
版 次：2002年2月第1版 2016年9月第4版 **印 次**：2018年1月第5次印刷
印 数：20001～23000
定 价：44.80元

产品编号：068610-01

出版说明

在教育部关于高等学校计算机基础教育三层次方案的指导下，我国高等学校的计算机基础教育事业蓬勃发展。经过多年的教学改革与实践，全国很多学校在计算机基础教育这一领域中积累了大量宝贵的经验，取得了许多可喜的成果。

随着科教兴国战略的实施以及社会信息化进程的加快，目前我国的高等教育事业正面临着新的发展机遇，但同时也必须面对新的挑战，这些都对高等学校的计算机基础教育提出了更高的要求。为了适应教学改革的需要，进一步推动我国高等学校计算机基础教育事业的发展，我们在全国各高等学校精心挖掘和遴选了一批经过教学实践检验的优秀的教学成果，编辑出版了这套教材。教材的选题范围涵盖了计算机基础教育的三个层次，包括面向各高校开设的计算机必修课、选修课，以及与各类专业相结合的计算机课程。

为了保证出版质量，同时更好地适应教学需求，本套教材将采取开放的体系和滚动出版的方式（即成熟一本、出版一本，并保持不断更新）。坚持宁缺毋滥的原则，力求反映我国高等学校计算机基础教育的最新成果，使本套丛书无论在技术质量上还是文字质量上均成为真正的“精选”。

清华大学出版社一直致力于计算机教育用书的出版工作，在计算机基础教育领域出版了许多优秀的教材。本套教材的出版将进一步丰富和扩大我社在这一领域的选题范围、层次和深度，以适应高校计算机基础教育课程层次化、多样化的趋势，从而更好地满足各学校由于条件、师资和生源水平、专业领域等的差异而产生的不同需求。我们热切期望全国广大教师能够积极参与到本套丛书的编写工作中来，把自己的教学成果与全国的同行们分享；同时也欢迎广大读者对本套教材提出宝贵意见，以便我们改进工作，为读者提供更好的服务。

我们的电子邮件地址是 jiaoh@tup.tsinghua.edu.cn。联系人：焦虹。

清华大学出版社

第 4 版前言

本书第 3 版自 2011 年推出已近 5 年。本次修订,除更新了部分计算机硬件发展新技术的描述之外,首次采用了以案例贯穿内容的方法。

计算机的主要应用方向之一是过程控制。工业过程控制的对象往往是一些连续变化的非电物理量,要使这类信号能够被计算机所识别和处理,需要经过一个复杂的过程。作为以理工类专业学生或即将从事计算机过程控制系统设计的学习者为目标读者的教材,本书引入了一个模拟的“家庭安全防盗系统”案例,贯穿到全书。从第 1 章的基础知识,到处理器工作原理、指令集合汇编程序设计方法,再到存储器技术、I/O 接口技术,最终完成对“家庭安全防盗系统”的设计。

当然,现实中设计这样一个系统使用单片机技术会更加适合,但考虑到人们日常见到和使用最多的计算机是微型计算机,建立微型计算机系统的整体概念,理解微型计算机的构成、工作原理、输入输出控制方法等,具有更普适的意义。因此,本书还是“奢侈”地选择了微型计算机来完成这一案例的设计。事实上,虽然单片机在体系结构、指令集等多个方面与微型计算机都存在较大差异,但它依然可以说是计算机的“微缩版”。理解了本书所介绍的内容,将非常有助于进一步理解单片机技术、嵌入式技术等。

5 年来,虽然微型计算机技术又有了飞速的发展,但其基本工作原理和基本体系结构依然是冯·诺依曼结构,作为介绍微型机工作原理的书籍,第 3 版中的大多数内容依然适用。因此,本次修订对原书中多数内容依然保留,面向应用、与实际工程设计相结合的特点没有改变,文字的叙述风格也一如既往。

乔亚男参与了本书第 1、2 章的修订。其余内容由吴宁修订。本次修订得到了我校陈文革老师的帮助和指导,借此深表谢意。

由于是首次引入案例描述方法,加之时间较紧,编者水平有限,因此还有进一步完善的空间,敬请同行和各位读者批评指正。

编　者

第3版前言

本书第2版推出已经3年。3年中,有关微型计算机的新技术在不断出现,微机的性能在不断提高,教学中也不断地有新的体会,这些都促使本书必须做适当的更新和调整。此次再版,原书中多数内容依然保留,面向应用、与实际工程设计相结合的特点没有改变,文字的叙述风格也一如既往。保留这些主要是基于以下理由。

虽然Intel公司的微处理器从早期的8086、8088、80286、80386,到后来的Pentium Pro、PentiumⅡ、PentiumⅢ、Pentium 4,再到今天的多核技术,无论是制造工艺还是技术和性能,都有了极大的改进和提高,但从应用者的角度,特别是应用程序员的角度来看,它们属于一个系列,是完全兼容的:应用编程的寄存器结构只有字长之分,而无本质区别;芯片的指令系统中,从8086到Pentium系列,除部分保护模式下扩展的指令外,80%以上是完全相同的;在应用程序中所用到的绝大多数指令依然是基本指令集中的指令,也就是8086指令集。因此,本书此次仍保留了第2版教材中关于微处理器的内容,包括Intel公司3个不同时期的代表性芯片8088、80386和Pentium 4的介绍,指令系统仍然以介绍8086指令集为主。

如今,虽然微型机的存储器容量越来越大,但存储器的基本工作原理和构成没有变;虽然微型机能够连接的外部设备越来越丰富,但中断工作原理和输入输出控制方法没有变;虽然随着大规模集成电路技术的发展,主板上曾经大量独立的接口芯片都已被集成到几块专用芯片中,但并行接口8255、串行接口8250、定时/计数器8253(8254)、中断控制器8259A等芯片的作用依然存在。

总之,虽然微型计算机技术有了巨大的发展,但其基本工作原理是相同的。作为介绍微型机工作原理的书籍,原版中的大多数内容依然适用。读者在学习时应以了解和掌握微型机的基本工作原理及应用方法为主,对部分复杂的新技术可根据需要参考其他相关的专业书籍。

本次修订得到了我校陈文革老师的大力支持和指导,在此表示诚挚的感谢。由于编者水平有限,书中难免存在一些疏漏和不当之处,敬请同行和各位读者批评指正。

编　者

目录



第1章 微型计算机基础概论

引言：

完成家庭安全防盗系统设计，首先需要了解微型计算机系统的组成以及计算机中的信息表示方法。本章主要介绍这两方面的内容。包括：

(1) 微型计算机系统，包括微型计算机的发展历程、微机系统的组成及各部分的主要功能。这样安排的目的是帮助读者首先建立起微机系统，特别是微型计算机硬件系统的整体概念，以便在后续章节的学习中始终能够有一个整体的结构框架。

(2) 计算机中的数制及编码的表示方法、它们相互间的转换、二进制数的运算、定点数和浮点数的表示等。这些都属于计算机基础知识。

教学目的：

- 理解微机系统的整体结构；
- 掌握三种常用记数制、两种编码的表示方法及其相互间的转换；
- 掌握二进制数的算术运算和逻辑运算；
- 深入理解补码的概念及其运算。

计算机的主要应用方向之一是过程控制。工业中的过程控制是指以温度、压力、流量等工艺参数作为被控变量的自动控制。这些被控变量通常是连续变化的非电物理量，但计算机只能处理离散电信号，对这类既非离散又非电信号的变量，如何进行控制呢？这需要一个“长长的处理过程”。

案例：随着社会的进步和经济的发展，二十多年来，人们的生活水平和生活环境都有了极大的改善，对家庭安全防盗措施也提出了新的要求。现有某住户需要设计一套家庭式电子安全防盗系统。该住户的住宅包括4间卧室、2间客厅、1间厨房和2个卫生间。其中，除一个卫生间无窗外，其他所有房间都含一扇可开关的窗，即共有8个窗户。系统的总体设计要求是：

(1) 为每个窗台安装监测装置，当出现异常时，启动报警(警铃响，警灯闪烁)，并在危险解除后关闭报警；

(2) 当住户外出或需要时使安全防盗系统处于布防状态，在不需要时则可关闭系统；

(3) 对异常的监测方法可以定时循环检测，也可以始终处于监测状态。

要完成这样一个系统设计，需要知道：

(1) 如何才能监测到异常以及异常信息如何才能被计算机所感知？

(2) 异常或正常信息在计算机中如何表示？如何存储？

(3) 计算机如何确定所接收到的来自监测设备的信息是正常还是异常？

(4) 对接收到的信息如何处理？如何发出报警信息？等等。

整个系统涉及硬联线路(硬件)设计和控制程序(软件)设计两大部分,以及一些计算机的基础知识,而这些就是本书所要介绍的内容。学习完本书,就可以完成这样一个过程控制系统核心部分的设计了。

在正式学习之前,有几点需要声明一下：一是关于软件设计。本书介绍汇编语言的目的并不是要求读者一定要使用汇编语言设计过程控制程序(目前更多情况下会使用 C 语言等高级语言),而是学习汇编语言更有助于对微型计算机工作原理的理解;第二,虽然书中作为案例介绍的芯片型号都显得有些"古老",但从应用的角度,其基本功能和使用方法与今天的新型器件是类似的。掌握了基本知识,也就具备了从事相关系统设计的基础;第三,设计这样一个家庭安全防盗系统,利用今天的微机系统进行控制成本有点高了,使用单片机技术实现会更加适合。什么是单片机呢？可以简单地说,单片机是计算机系统的"微缩版",虽然它与计算机在体系结构、指令集等多个方面都存在较大差异,但它内部包括了计算机的主要功能部件,如 CPU、内存、总线、存储器、接口等,只是这些部件的性能相对微型计算机要弱很多。

由于人们日常见到和使用最多的计算机是微型计算机,建立"微机系统"的整体概念,理解微型计算机的构成、工作原理、输入输出控制方法等,具有更普适的意义,因此,本书还是"奢侈"地以微型计算机为例,来完成上述案例的设计。

1.1 微型计算机系统

本节概述微型计算机的发展历程、微机的一般工作过程以及微机系统的组成 3 个方面的内容。

微型计算机的发展更替主要是指微处理器的更新换代。微处理器发展的重要基础是电子技术的发展,中间复杂的原理这里不做讨论,只简单地说明一下它们各自的特点。

事实上,微型计算机的工作原理只有在学习完这本书后才能完全明白。1.1 节只是以流程图和框图的形式简单说明微机的一般工作过程。

本书讨论的对象是微型计算机的硬件系统。在进一步学习硬件各部分的详细构成和工作原理之前,先建立起整个系统的概念是必要的。1.1.3 节将通过结构框图介绍微机系统的概念结构和层次结构。

1.1.1 微型计算机的发展

计算机技术是 20 世纪发展最快的技术之一。自 1946 年第一台计算机问世以来,在短短的六十多年中,已经历了由电子管计算机、晶体管计算机、集成电路计算机到大规模、超大规模集成电路计算机这样五代的更替,并且还在不断地向巨型化、微型化、网络化和

智能化这 4 个方向发展。

计算机按照性能、价格和体积等的综合指标，可分为巨型机、大型机、中型机、小型机、微型机五大类。

微型计算机诞生于 20 世纪 70 年代，由于体积小、价格低，尤其是日益提高的性能价格比，使其迅速在各行各业乃至家庭中得到了广泛的应用。现在一台微型机的处理能力不仅早已超过了 20 世纪 50 年代初期占地上千平方英尺、重量数十吨、功耗几百千瓦的大型电子管计算机，而且大大超过了二十多年前、造价数十万美元的大型晶体管数字计算机。

微处理器是微型计算机的核心芯片，简称 μP 或 MP(Micro Processor)。它将计算机中的运算器和控制器集成在一片硅片上，也称为中央处理单元，即 CPU（Central Processing Unit）。它是 20 世纪 70 年代人类重要的创新之一，在四十多年的时间中获得了极快的发展，其集成度和性能几乎每过一年就会提高 1.5～2 倍。

微处理器和微型计算机的发展历史是与大规模集成电路的发展分不开的。20 世纪 60 年代初期的硅平面管工艺和二极管晶体管逻辑电路的发展，使得在 1963 年、1964 年有了小规模集成电路（Small Scale Integration，SSI）的出现，之后的金属氧化物半导体（Metal Oxide Semiconductor，MOS）工艺，又使集成度提高了一大步。到 20 世纪 60 年代后期，在一片几平方毫米的硅片上，已可集成几千个晶体管，这就出现了大规模集成电路 LSI（Large Scale Integration）。LSI 器件体积小、功耗低、可靠性高，为微处理器的生产打下了很好基础。现代新型的集成电路已可在单个芯片上集成数亿个晶体管，工作频率超过 3GHz。

虽然集成电路技术在不断发展，但终归受物理性能的限制而存在极限。微处理器的两大生产巨头 Intel 和 AMD 发现：单纯地通过提高芯片的集成度以提升工作频率，已无法明显提升系统整体性能，性能的提高会受到多种因素的制约：处理器内部的计算速度和外部访问存储器的访问速度的差异越来越大，由于访存的限制，使得处理器的性能很难再有明显的提高；随着功率的增大，散热问题成为了一个无法逾越的障碍；超标量和超流水线技术已接近了极限；开发成本也在不断提高。于是，到了 2004 年左右，尽管晶体管数目还是呈线性增加，但时钟频率和性能都已达到拐点，按照传统的提高芯片时钟频率的方式来提高系统的性能已经走到了尽头。在这个背景下，片上多核处理器（Chip of Multiprocessor，CMP）技术应运而生。

从微处理器诞生到 20 世纪末，每块处理器中都只有一个“核心”，称为单核处理器。这里的核心又称为内核，是 CPU 最重要的组成部分，由单晶硅以一定的生产工艺制造出来。CPU 所有的计算、数据处理、接受和存储命令都由核心执行。而“多核”处理器技术试图通过增加 CPU 上的核心数量来突破主频限制、提高性能。简单地说，就是将多个功能相同的计算内核集成在一个处理器中，使处理器每个时钟周期内的执行能力随着计算内核的个数增加而大幅度增加，从而提高了计算能力。

IBM 于 2001 年发布了第一款多核处理器 POWER4；紧接着，AMD 和 Intel 也都于 2005 年前后推出了自己的首款多核处理器芯片 AMD Opteron 和 Core Duo。经过十几年时间的发展，如今市场上已经有大量多核处理器芯片可供选择，如 Intel Haswell、Intel

Xeon Phi、AMD Cortex-A9、Nvidia GPGPU 等。由于目前对于多核处理器的设计还没有完全统一的标准，因此各大厂商多核处理器的设计目标也会有所不同：Intel 仍然是以强化单个处理器核的计算性能和效率为目标，主要关注于高性能计算领域，设计复杂的处理器核以最大化单线程的性能；AMD 主要注重整个多核处理器系统的任务吞吐量，简化了单个处理器核的设计结构，融入了更多的处理器核并强化处围部件的结构设计；ARM 的设计目标则是低功耗、高性能和低成本，主要关注嵌入式系统和移动通信领域；Nvidia 则是以最大化芯片吞吐量为设计目标，最大程度地提高可集成处理器核的数量；IBM 则专注于高性能服务器市场，最大程度地挖掘所有可用资源，提高系统整体的运行效率，如 IBM Power 7＋最高主频达到了 5.5GHz，末级 Cache 容量也达到了 80MB。

1.1.2 微型计算机的工作过程

1. 冯·诺依曼计算机

计算机的工作过程就是执行程序的过程，而程序则是指令序列的集合。那么，什么是指令呢？其实，指令可以说就是人向计算机发出的、能够被计算机所识别的命令。不同型号的计算机（准确地说应是处理器）识别“命令”的能力不同，即其能够执行的指令不同。人们将计算机所能够识别的所有指令的集合称为该机的指令系统。本书的第 3 章将详细介绍 Intel 80x86 CPU 的指令系统。

当人们要利用计算机完成某项工作，例如，要解算一道数学题时，需要先把题目的解算方法分解成计算机能够识别并能执行的基本操作命令。这些基本操作命令按一定顺序排列起来，就组成了程序，而其中每一条基本操作命令称为一条机器指令，指示计算机执行规定的操作。

因此，程序是实现既定任务的指令序列，计算机按照程序安排的顺序执行指令，就可完成解题任务。

每台计算机都拥有各种类型的机器指令，这些指令按照一定的规则存放在存储器中，在中央控制系统的统一控制下，按一定顺序依次取出执行，这就是冯·诺依曼计算机的核心原理，即存储程序的工作原理。存储程序的概念是指把程序和数据送到具有记忆功能的存储器中保存起来，计算机工作时只要给出程序中第一条指令的地址，控制器就可依据存储程序中的指令顺序地、周而复始地取出指令、分析指令、执行指令，直到执行完全部指令为止。

冯·诺依曼计算机的主要特点如下。

（1）将计算过程描述为由许多条指令按一定顺序组成的程序，并放入存储器保存。

（2）程序中的指令和数据必须采用二进制编码，且能够被执行该程序的计算机所识别。

（3）指令按其在存储器中存放的顺序执行，存储器的字长固定并按顺序线性编址。

（4）由控制器控制整个程序和数据的存取以及程序的执行。

（5）以运算器为核心，所有的执行都经过运算器。

多年来，尽管计算机体系结构发生了重大变化、性能不断改进提高，但从本质上讲，存

储程序控制仍是现代计算机的结构基础。图 1-1 是典型的冯·诺依曼计算机结构示意图，其各部分的职责和功能本书将在后续章节中详细介绍。

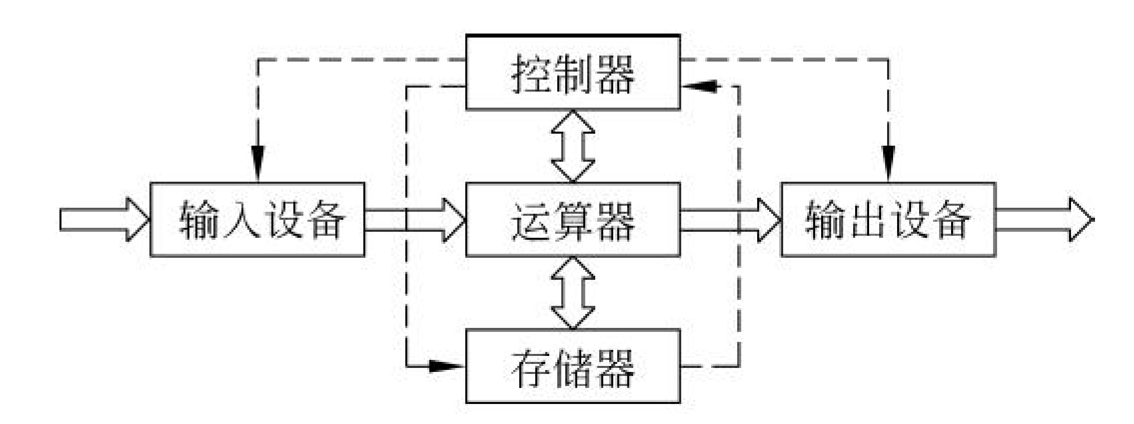


图 1-1　冯·诺依曼计算机结构示意图

2. 微型计算机的工作过程

如上所述，微机的工作过程就是执行程序的过程，也就是逐条执行指令序列的过程。由于每一条指令的执行都包括取指令和执行指令两个基本阶段，所以微机的工作过程也就是不断地取指令和执行指令的过程。图 1-2 是执行这个过程的示意图。

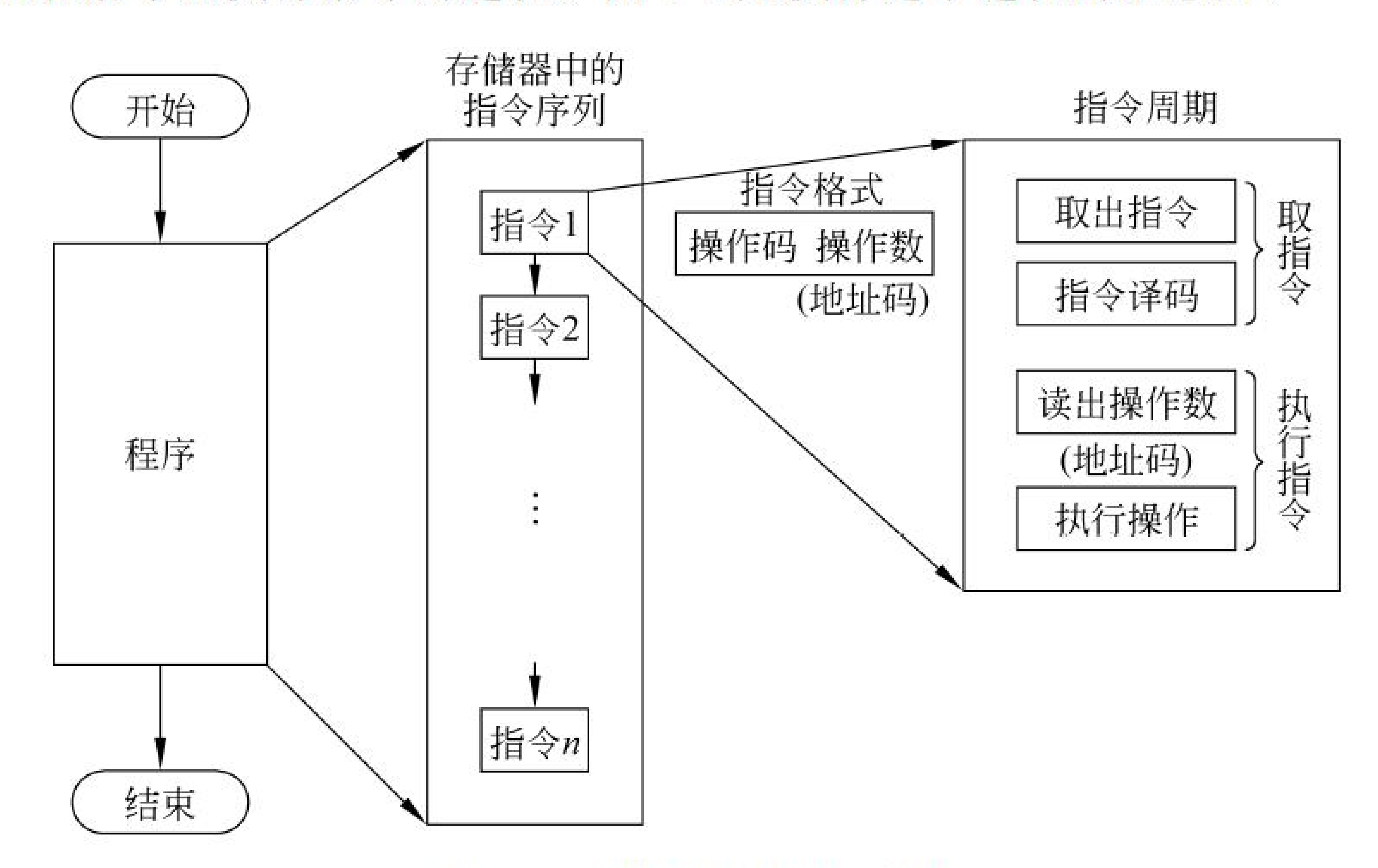


图 1-2　程序执行过程示意图

假定程序已由输入设备存放到内存中，当计算机要从停机状态进入运行状态时：

① 首先将第一条指令由内存中取出；

② 将取出的指令送指令译码器译码，以确定要进行的操作；

③ 读取相应的操作数(即执行的对象)；

④ 执行指令；

⑤ 存放执行结果；

⑥ 一条指令执行完后，转入了下一条指令的取指令阶段，如此周而复始地循环，直到程序中遇到暂停指令方才结束。

取指令阶段都是由一系列相同的操作组成的，所以取指令阶段的时间总是相同的，称为公共操作。而执行阶段则由不同的事件顺序组成，它取决于被执行指令的类型。因此，指令不同，执行阶段所花费的时间也各不相同。

图 1-3 是一个简单实例中读取第一条指令的工作过程示意图。

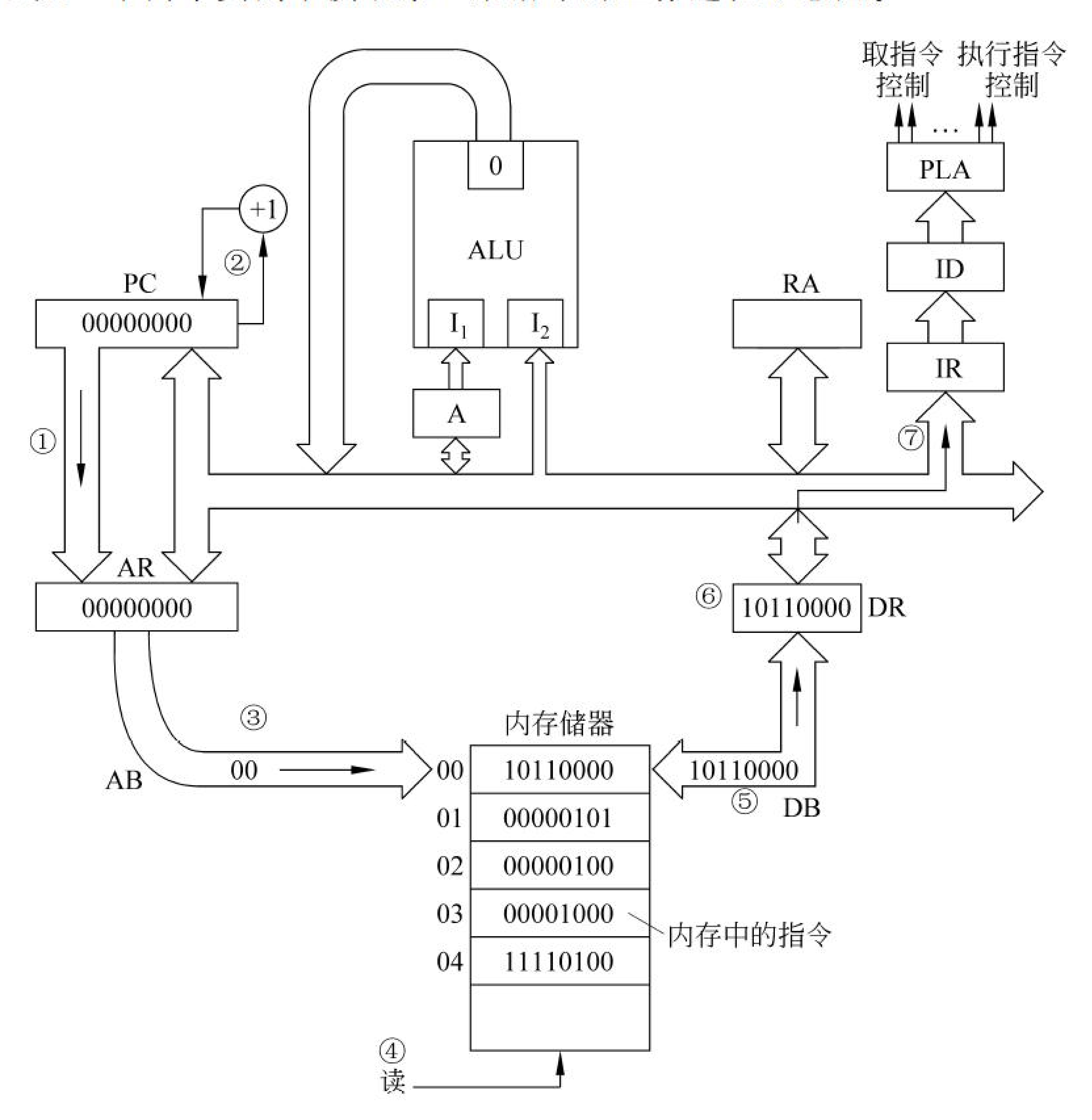


图 1-3　读取第 1 条指令操作码的过程

编写一个求 5＋8＝？的程序。其机器码和助记符程序如下。

机器码	助记符	
10110000 00000101	MOV A,5	;第 1个操作数 (5)送到累加器
00000100 00001000	ADD A,8	;5与第 2个数 (8)相加,结果 (13)送到累加器
11110100	HLT	;停机

取第一条指令的过程如下。

(1) 指令所在的地址(这里为 00000000)赋给程序计数器 PC 并送到地址寄存器 AR。

(2) PC 自动加 1(即由 00000000 变为 00000001),AR 的内容不变。

(3) 将地址寄存器 AR 的内容(00000000)放在地址总线上,并送至内存储器,经地址译码器译码,选中相应的 00000000 单元。

(4) CPU 的控制器发出读命令。

(5) 在读命令控制下,将所选中的 00000000 单元中的内容即第 1 条指令的操作码 10110000 读到数据总线 DB。

(6) 将读出的内容 10110000 经数据总线送到数据寄存器 DR。

(7) 取指令阶段的最后一步是指令译码。因为取出的是指令的操作码,故数据寄存

器 DR 将它送到指令寄存器 IR,然后再送到指令译码器 ID。

如此,就完成了第 1 条指令的读取。第 2 条以及后续指令的读取过程与第 1 条指令是一样的,只是每次译码后指令译码器 ID 中的内容不同(因为指令不同)。

1.1.3 微机系统的组成

微型计算机(Microcomputer)是体积、重量、计算能力都相对比较小的一类计算机的总称,一般供个人使用,所以也称为个人计算机(Personal Computer,PC)。

人们通常所说的微型机实际上指的是微型机系统。微机系统、微型机和微处理器是 3 个不同的概念,是微型计算机从全局到局部的 3 个不同的层次。微型计算机系统的概念结构如图 1-4 所示,它由硬件系统和软件系统两大部分组成。

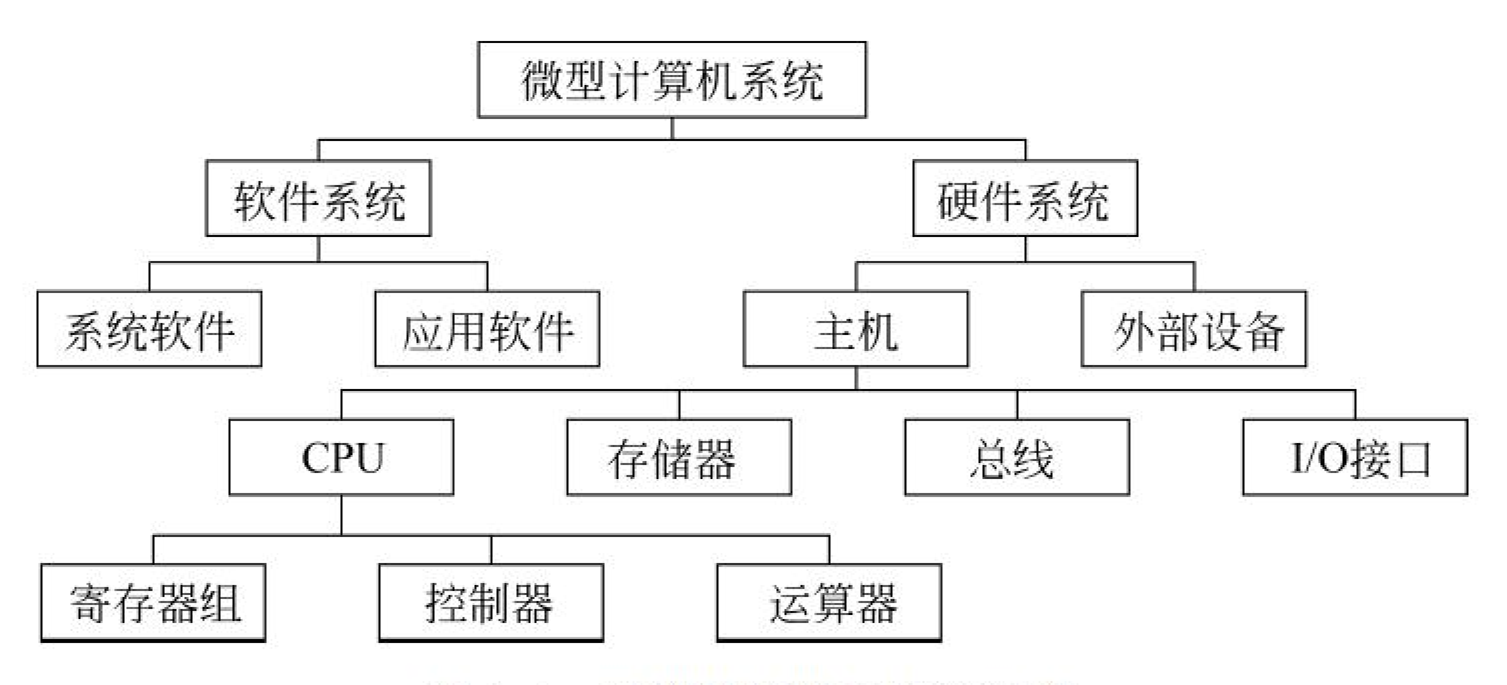


图 1-4 微型计算机系统的组成

对于硬件系统,目前的各种微型计算机,无论是单片机还是个人计算机系统,从概念结构上来说都是由微处理器、存储器及输入输出接口等几个部分组成。在具体实现上,这些组成部分往往又合并或分解为若干个功能模块,分别由不同的部件予以实现。各组成部分之间通过总线连接,总线是部件之间信息传递的公共通道。所以通常也将总线系统作为硬件主机系统的一个独立部件。

所有的微型机系统都采用了总线结构形式。总线结构的主要优点是设计简单、灵活性好、具有优良的可扩展性、便于故障检测和维修。图 1-5 为微型计算机的系统结构框图。图中 AB 表示地址总线(Address Bus),用于传送读/写存储器(RAM 或 ROM)或读/写输入输出接口(I/O 接口)的地址信息;DB 表示数据总线(Data Bus),传送操作的数据;CB 表示控制总线(Control Bus),传送控制信息。

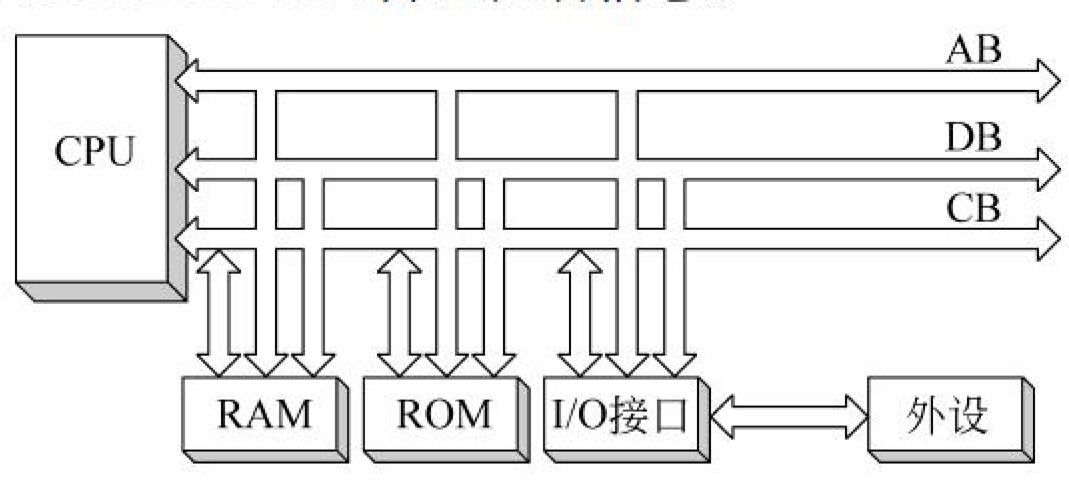


图 1-5 微型计算机的系统结构框图

1. 硬件系统

1）微处理器（或中央处理器、CPU）

CPU 是微型计算机的核心芯片，是整个系统的运算和指挥控制中心。不同型号的微型计算机，其性能的差别首先在于其 CPU 性能的不同，而 CPU 性能又与它的内部结构有关。无论哪种 CPU，其内部基本组成都大同小异，即包括控制器、运算器和寄存器组 3 个主要部分。CPU 的典型结构如图 1-6 所示。

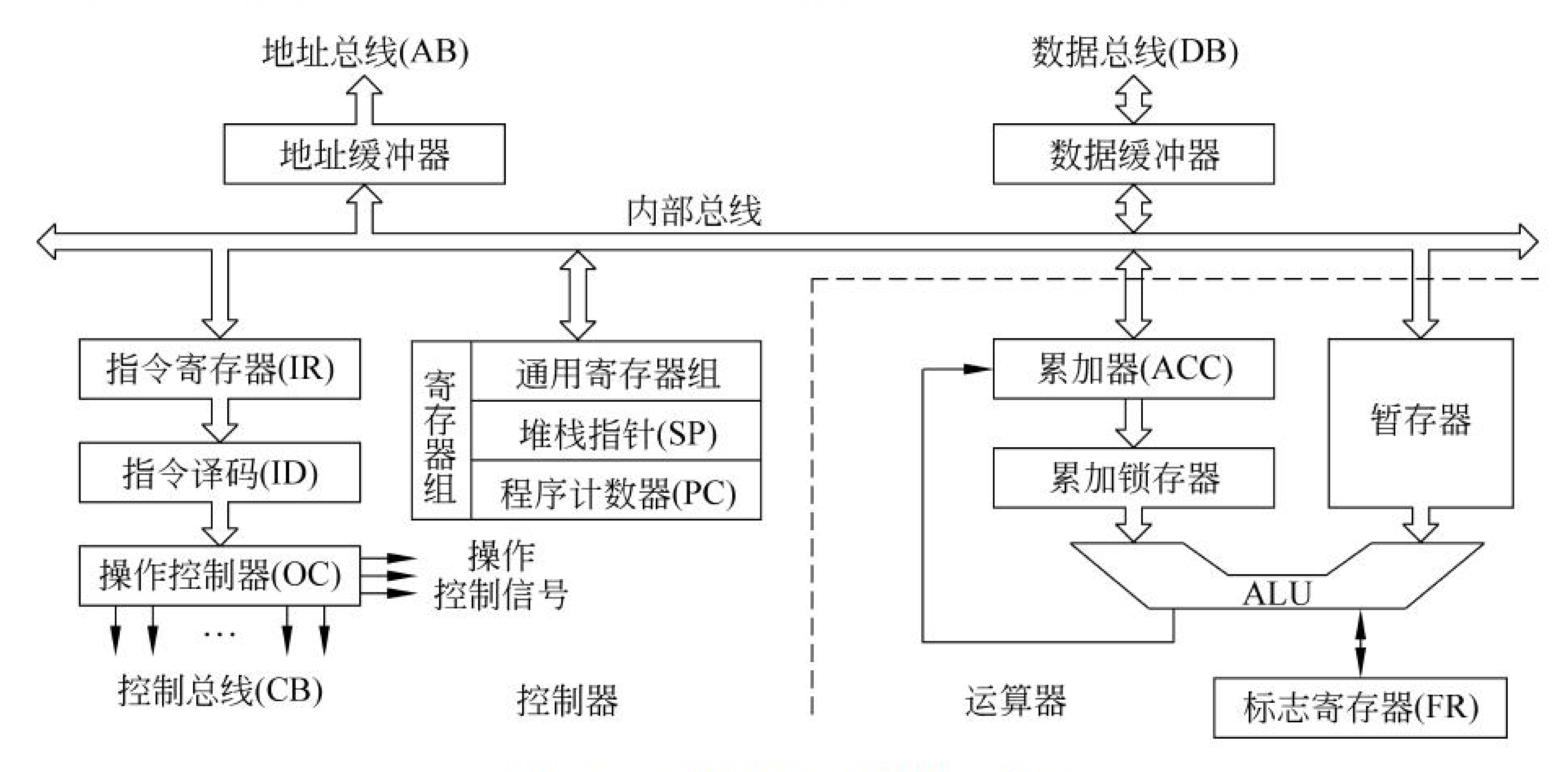


图 1-6　微处理器典型结构示意图

(1) 运算器：运算器的核心部件是算术逻辑单元（ALU，Arithmetic and Logic Unit），它是以加法器为基础，辅之以移位寄存器及相应控制逻辑组合而成的电路，在控制信号的作用下可完成加、减、乘、除四则运算和各种逻辑运算。现代新型 CPU 的运算器还可完成各种浮点运算。

(2) 控制器：一般由指令寄存器、指令译码器和操作控制电路组成。控制器是整个 CPU 的指挥控制中心，对协调整个微型计算机有序工作极为重要。它从存储器中依次取出程序的各条指令，并根据指令的要求，向微机的各个部件发出相应的控制信号，使各部件协调工作，从而实现对整个微机系统的控制。

(3) 寄存器组：实质上是 CPU 内部的若干个存储单元，在汇编语言中通常是按名字来访问它们。寄存器可分为专用寄存器和通用寄存器。专用寄存器的作用是固定的，如堆栈指针、程序计数器、标志寄存器等。而通用寄存器则可由程序员规定其用途。通用寄存器的数目因 CPU 而异，如 8088/8086 CPU 中就有 8 个 16 位通用寄存器可供程序员使用。由于有了这些寄存器，在需要重复使用某些操作数或中间结果时，就可将它们暂时存放在寄存器中，避免对存储器的频繁访问，从而缩短指令长度和指令执行时间，同时也给编程带来很大的方便。

除了上述两类程序员可用的寄存器外，微处理器中还有一些不能直接为程序员所用的寄存器，如累加锁存器、暂存器和指令寄存器等，它们仅受内部定时与控制逻辑的控制。

有关微处理器的具体的结构和工作原理将在本书第 2 章做进一步讨论。

2）存储器

主机系统中的存储器(Memory)又叫内存或主存，是微型计算机的存储和记忆部件，用以存放数据(包括原始数据、中间结果和最终结果)和当前执行的程序。微型机的内存均由半导体材料制成，故也称半导体存储器。

(1) 内存单元的地址和内容。内存由许多单元组成，每个单元可存放一组二进制码。在微型机中，每个内存单元规定存放 8 位二进制数，即一个字节(8b)。一台微机中内存单元的总数称为该微机的内存容量，单位为字节。例如，一台微机拥有 4×2^{20} 个内存单元，就称该微机的内存容量为 4MB。为了区分各个不同的内存单元，需要给每个存储单元编上不同的号码，这个编号称为内存地址。内存地址编号从 0 开始顺序编排。例如，8088/8086 CPU 的内存地址编码为 00000H、00001H、…、FFFFFH[①]，共 2^{20} 个存储单元。因为每个存储单元都有一个唯一的地址，所以 CPU 要访问某个内存单元时，就可以通过指定该内存单元的地址来正确地访问它。

内存单元中存放的信息称为内存单元的内容。虽然内存单元的内容与内存单元的地址在表现形式上都是二进制数，但本质上它们是两个完全不同的概念。图 1-7 给出了这两个概念的示意图。图中，地址为 F0000H 的存储单元中存放的内容为 00111110B(或 3EH)，记为(F0000H)＝3EH。

地址	内容
00001H	11000111
00002H	00001100
	⋮
F0000H	00111110
	⋮
FFFFFH	01110010

图 1-7　内存单元的地址和内容

(2) 内存的操作。CPU 对内存的操作有读、写两种。读操作是 CPU 将内存单元的内容取到 CPU 内部，而写操作是 CPU 将其内部信息传送到内存单元保存起来。显然，写操作的结果改变了被写单元的内容，而读操作则不改变被读单元的内容。

现假定存储器由 256 个单元组成，地址从 00H～FFH，每个单元存储 8 位二进制信息，即字长为 8 位，其结构简图如图 1-7 所示。这种规格的存储器通常被称为容量为 256 字节的读写存储器。

从存储器读出信息的操作过程如图 1-8(a)所示。CPU 读出地址为 04H 内存单元中的内容的过程如下。

① CPU 把地址 04H 放到地址总线上，经地址译码器选中 04H 单元。

② CPU 发出“读”控制信号。

③ 读出存储器 04H 号单元中的内容 97H(10010111B)并送到数据总线上。

应当指出，读操作完成后，04H 单元中的内容 97H 仍保持不变，这种特点称为非破坏性读出(Non Destructive Read Out)。这一特点很重要，因为它允许多次从某个存储单元读出同一备份。

向存储器写入信息的操作过程如图 1-8(b)所示。假定 CPU 要把数据 00100110B(26H)写入地址为 08H 的存储单元，则步骤如下。

① CPU 将存储单元地址 08H 放到地址总线上，经地址译码器选中 08H 单元。

② CPU 将要写入的内容 26H 放到数据总线上。

---

① 这里的“H”是十六进制数的标识符，而下文中的“B”则是二进制数的标识符。

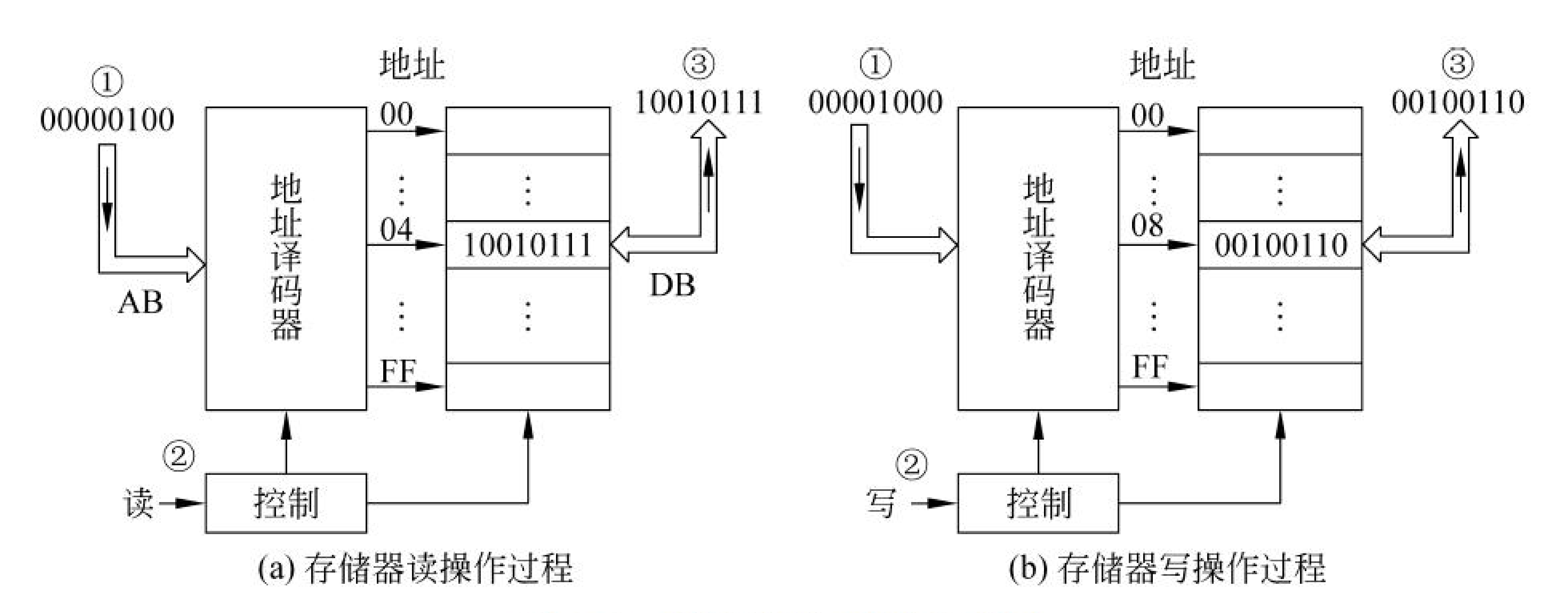


图 1-8　存储器读写操作示意图

③ CPU 向存储器发送“写”控制信号,在该信号的控制下,将数据 26H 写入存储器的 08H 单元。

应当注意,写入操作将破坏该单元原存的内容,即由新内容 26H 代替了原存内容,原存内容将被清除。

上述类型的存储器称为随机存取存储器,既可以读出也可以写入信息。

(3) 内存的分类。按工作方式不同,内存可分为两大类:随机存取存储器 RAM (Random Access Memory)和只读存储器 ROM(Read Only Memory)。

RAM 可以被 CPU 随机地读和写,所以又称为读写存储器。这种存储器用于存放用户装入的程序、数据及部分系统信息。当机器断电后,所存信息消失。

ROM 中的信息只能被 CPU 随机读取,而不能由 CPU 任意写入。机器断电后,信息并不丢失。所以,这种存储器主要用来存放监控程序和基本输入输出程序,还可用来存放各种常用数据和表格等。ROM 中的内容一般是由生产厂家或用户使用专用设备写入固化的。

有关存储器的详细内容将在本书第 5 章中详细叙述。

3) 输入输出接口和输入输出设备

输入输出(I/O)设备和输入输出接口是输入输出系统的硬件组成,而 I/O 系统是微型计算机系统的重要组成部分。

I/O 设备中,常用的输入设备有键盘、鼠标器、扫描仪等;常用的输出设备有显示器、打印机、绘图仪等。磁带、磁盘等既是输入设备,又是输出设备。

I/O 设备的种类繁多,结构、原理各异,有机械式、电子式、电磁式等。与 CPU 相比,I/O 设备的工作速度较低,处理的信息从数据格式到逻辑时序一般都不可能与计算机直接兼容。因此,微机与 I/O 设备间的连接与信息交换不能直接进行,而必须通过一个中间部件作为两者之间的桥梁,该部件就叫做输入输出接口(I/O 接口)。I/O 接口有时又称为 I/O 适配器(I/O Adapter)。

有关输入输出系统,特别是输入输出接口的概念和应用将在本书第 6 章中详细介绍。

4) 总线

总线(Bus)由一组导线和相关控制电路组成,是各种公共信号线的集合,用于微机系统各部件之间的信息传递。通常将用于主机系统内部信息传递的总线称为内部总线,将

连接主机和外部设备之间的总线称为外部总线。从传送信息的类型上，这两类总线都包括用于传送数据的数据总线、传送地址信息的地址总线和传送控制信息的控制总线。

(1) 数据总线 DB(Data Bus)。数据总线用来传输数据信息，是双向总线，CPU 既可通过 DB 从内存或输入设备输入数据，也可通过 DB 将内部数据送至内存或输出设备。

(2) 地址总线 AB(Address Bus)。地址总线用于传送 CPU 发出的地址信息，是单向总线。传送地址信息的目的是指明与 CPU 交换信息的内存单元或 I/O 设备。

(3) 控制总线 CB(Control Bus)。控制总线用来传送控制信号、时序信号和状态信息等。其中有的是 CPU 向内存和外设发出的信息，有的则是内存或外设向 CPU 发出的信息。可见，CB 中每一根线的方向是一定的、单向的，但 CB 作为一个整体是双向的。所以在各种结构图中凡涉及控制总线 CB，均以双向线表示。

对总线系统的进一步了解参见本书第 2 章。

2. 软件系统

软件包括系统软件和应用软件两大类。

应用软件是用户为解决各种实际问题(如数学计算、检测与实时控制、音乐播放等)而编制的程序。

系统软件主要包括操作系统(OS)和系统实用程序。操作系统是一套复杂的系统程序，用于管理计算机的硬件与软件资源、进行任务调度、提供文件管理系统、人机接口等。操作系统还包含了各种 I/O 设备的驱动程序。

系统实用程序包括各种高级语言的翻译/编译程序、汇编程序、数据库系统、文本编辑程序以及诊断和调试程序，此外还包括许多系统工具程序等。

计算机中的程序设计语言分为 3 个级别，第一级是机器语言，第二级是汇编语言，第三级是高级语言。机器语言程序是计算机能理解和直接执行的二进制形式的程序。汇编语言程序是用助记符语言表示的程序，计算机不能直接“识别”，需经过“汇编程序”的翻译把它转换为机器语言方能执行。机器语言指令与汇编语言指令基本上一一对应，都是与硬件密切相关的。而高级语言是不依赖于具体机型的程序设计语言，由它所编写的程序需经过编译程序或解释程序的翻译方能执行。

文本编辑程序是供输入或修改文本(字母、数字和标点等组成的一组字符或代码序列)用的程序，它可用来输入、编辑源程序，当然也可编辑文章。

在编写程序时，还可能需要另外 3 种系统程序：系统程序库、连接程序与装入程序。

一般操作系统都有一个通用的系统程序库，用户还可以建立自己的程序库(一组子程序)。程序库中的子程序可附在任何系统程序或用户程序上以供调用。把待执行的程序与程序库及其他已翻译好的程序连接起来成为一个整体的准备程序称为连接程序。另一种准备程序是用来把待执行的程序加载到内存中，称为装入程序。有时，连接与装入功能可合成一个程序。

应当指出，硬件系统和软件系统是相辅相成的，共同构成微型计算机系统，缺一不可。现代的计算机硬件系统和软件系统之间的分界线并不是绝对的，总的趋势是两者统一融合，在发展上互相促进。

由于本书的宗旨是讨论有关计算机硬件技术方面的知识，对软件系统仅做简单介绍。现代计算机的程序设计中，多以高级语言进行，但高级语言程序与具体的硬件系统无关。为了真正理解微型机的工作过程，书中的程序设计均以汇编语言为主，基本不涉及高级语言。汇编语言的程序设计将在本书第 5 章中讨论。

1.2 计算机中的数制及编码

在日常生活中，人们习惯于使用十进制数来进行计数和计算。但现代数字计算机主要都是由开关元件构成，故只能识别由 0 和 1 构成的二进制代码，也就是说计算机中的数是用二进制表示的。但用二进制数表示一个较大的数时，既冗长又难以记忆，为了阅读和书写方便，或适应某些特殊场合的需要，在计算机中有时也采用十六进制数和十进制数。所以，在学习计算机原理之前，首先需要了解和掌握这 3 种常用记数制及其相互间的转换。

1.2.1 常用记数制

1. 十进制数

十进制数共有 0～9 十个数字符号，用符号 D 标识，无论数的大小，都可用这 10 个符号的组合来表示。一个任意十进制数都可用权展开式表示为

$$(D)_{10} = D_{n-1} \times 10^{n-1} + D_{n-2} \times 10^{n-2} + \cdots + D_1 \times 10^1 + D_0 \times 10^0 + D_{-1} \times 10^{-1} + \cdots + D_{-m} \times 10^{-m} = \sum_{i=-m}^{n-1} D_i \times 10^i \quad (1.1)$$

式中：D_i 是 D 的第 i 位的数码，可以是 0～9 十个符号中的任何一个；n 和 m 为正整数，n 表示小数点左边的位数，m 表示小数点右边的位数；10 为基数，10^i 称为十进制的权。

【例 1-1】 十进制数 3256.87 可表示为

$$(3256.87)_{10} = 3 \times 10^3 + 2 \times 10^2 + 5 \times 10^1 + 6 \times 10^0 + 8 \times 10^{-1} + 7 \times 10^{-2}$$

2. 二进制数

二进制数的每一位只取 0 和 1 两个数字符号，用符号 B 标识，遵循逢二进一的法则。一个二进制数 B 可用其权展开式表示为

$$(B)_2 = B_{n-1} \times 2^{n-1} + B_{n-2} \times 2^{n-2} + \cdots + B_0 \times 2^0 + B_{-1} \times 2^{-1} + \cdots + B_{-m} \times 2^{-m} = \sum_{i=-m}^{n-1} B_i \times 2^i \quad (1.2)$$

式中：B_i 只能取 1 或 0；2 为基数，2^i 为二进制的权；m、n 的含义与十进制表达式相同。为与其他进位记数制相区别，一个二进制数通常用下标 2 表示。

【例 1-2】 二进制数 1010.11 可表示为

$$(1010.11)_2 = 1 \times 2^3 + 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 + 1 \times 2^{-1} + 1 \times 2^{-2}$$

3. 十六进制数

十六进制数共有 16 个数字符号，0～9 及 A～F，用符号 H 标识，其计数规律为逢十六进一。一个十六进制数 H 也可用权展开式表示为

$$(H)_{16}=H_{n-1}\times 16^{n-1}+H_{n-2}\times 16^{n-2}+\cdots+H_0\times 16^0+H_{-1}\times 16^{-1}+\cdots+H_{-m}\times 16^{-m}$$

$$=\sum_{i=-m}^{n-1}H_i\times 16^i \qquad (1.3)$$

式中：H_i 的取值在 0～F 的范围内，16 为基数，16^i 为十六进制数的权；m、n 的含义与上面相同。十六进制数也可用下标 16 表示。

【例 1-3】 十六进制数 2AE.4H 可表示为

$$(2AE.4)_{16}=2\times 16^2+10\times 16^1+14\times 16^0+4\times 16^{-1}$$

二进制数与十六进制数之间存在有一种特殊关系，即 $2^4=16$，也就是说 1 位十六进制数恰好可用 4 位二进制数来表示，且它们之间的关系是唯一的。所以，在计算机应用中，虽然机器只能识别二进制数，但在数字的表达上更广泛地采用十六进制数。

计算机中常用的二进制数、十六进制数和十进制数之间的关系如表 1-1 所示。

表 1-1 数制对照表

十进制数	二进制数	十六进制数	十进制数	二进制数	十六进制数
0	0000	0	8	1000	8
1	0001	1	9	1001	9
2	0010	2	10	1010	A
3	0011	3	11	1011	B
4	0100	4	12	1100	C
5	0101	5	13	1101	D
6	0110	6	14	1110	E
7	0111	7	15	1111	F

4. 其他进制数

除以上介绍的二、十和十六进制 3 种常用的进位记数制外，计算机中还可能用到八进制数，有兴趣的读者可自行将其计数及表达方法进行归纳，这里就不再详细介绍了。下面给出任一进位制数的权展开式的一般形式。

一般地，对任意一个 K 进制数 S，都可用权展开式表示为

$$(S)_k=S_{n-1}\times K^{n-1}+S_{n-2}\times K^{n-2}+\cdots+S_0\times K^0+S_{-1}\times K^{-1}$$

$$+\cdots+S_{-m}\times K^{-m}=\sum_{i=-m}^{n-1}S_i\times K^i \qquad (1.4)$$

式中：S_i 是 S 的第 i 位的数码，可以是所选定的 K 个符号中的任何一个；n 和 m 的含义同上，K 为基数，K^i 称为 K 进制数的权。

除了用基数作为下标来表示数的进制外，通常在不同进制数的后面加上其标识字母 B、H、D 等来分别表示二进制数、十六进制数和十进制数，如 11000101B、2C0FH、1300D

等。在不至于混淆时，十进制数后面的 D 可以省略。

1.2.2 各种数制之间的转换

人类习惯的是十进制数，计算机采用的是二进制数，编写程序时为方便起见又多采用十六进制数，因此必然会产生在不同记数制之间进行转换的问题。

1. 非十进制数到十进制数的转换

非十进制数转换为十进制数的方法比较简单，只要将它们按相应的权表达式展开，再按十进制运算规则求和，即可得到它们对应的十进制数。

【例 1-4】 将二进制数 1101.101 转换为十进制数。

解：根据二进制数的权展开式，有

$$(1101.101)_2 = 1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3} = (13.625)_{10}$$

【例 1-5】 将十六进制数 64.CH 转换为十进制数。

解：根据十六进制数的权展开式，有

$$(64.C)_{16} = 6\times16^1+4\times16^0+C\times16^{-1} = 6\times16^1+4\times16^0+12\times16^{-1} = (100.75)_{10}$$

2. 十进制数转换为非十进制数

1）十进制数转换为二进制数

十进制数整数和小数部分应分别进行转换。整数部分转换为二进制数时采用“除 2 取余”的方法，即连续除 2 并取余数作为结果，直至商为 0，得到的余数从低位到高位依次排列即得到转换后二进制数的整数部分；对小数部分，则用“乘 2 取整”的方法，即对小数部分连续用 2 乘，以最先得到的乘积的整数部分为最高位，直至达到所要求的精度或小数部分为 0 为止（可以看出，转换的结果的整数和小数部分是从小数点开始分别向高位和低位逐步扩展）。

【例 1-6】 将十进制数 112.25 转换为等值的二进制数。

解：

整数部分	小数部分
112/2=56………余数=0（最低位）	0.25×2=0.5………整数=0（最高位）
56/2=28………余数=0	0.5×2=1.0………整数=1
28/2=14………余数=0	
14/2=7………余数=0	
7/2=3………余数=1	
3/2=1………余数=1	
1/2=0………余数=1	

从而得到转换结果$(112.25)_{10} = (1110000.01)_2$。

2）十进制数转换为十六进制数

与十进制数转换为二进制数的方法类似，整数部分按“除 16 取余”的方法进行，小数部分则“乘 16 取整”。

【例 1-7】 将十进制数 301.6875 转换为等值的十六进制数。

解：

整数部分	小数部分
301/16＝18………余数＝D	0.6875×16＝11.0000……… 整数＝$(11)_{10}$＝$(B)_{16}$
18/16＝1 …………余数＝2	
1/16＝0 …………余数＝1	

所以有 $(301.6875)_{10}$＝ $(12D.B)_{16}$。

也可将十进制数先转换为二进制数，再转换为十六进制数，在下面将会看到后者的转换是非常方便的。

3. 二进制数与十六进制数之间的转换

由于 2^4＝16，故 1 位十六进制数能够表示的数值恰好相当于 4 位二进制数能够表示的数值，这就使十六进制数与二进制数之间的转换变得非常容易。

将二进制数转换为十六进制数的方法是：从小数点开始分别向左和向右把整数和小数部分每 4 位分为一组。若整数最高位的一组不足 4 位，则在其左边补零；若小数最低位的一组不足 4 位，则在其右边补零。然后将每组二进制数用对应的十六进制数代替，则得到转换结果。

【例 1-8】 将二进制数 110100110.101101B 转换为十六进制数。

解：

二进制数	0001	1010	0110.	1011	0100
	↓	↓	↓	↓	↓
十六进制数	1	A	6.	B	4

所以有 $(110100110.101101)_2$＝1A6.B4H。

十六进制数转换为二进制数的方法与上述过程相反，即用 4 位二进制代码取代对应的 1 位十六进制数。

【例 1-9】 将十六进制数 2A8F.6DH 转换为二进制数。

解：

十六进制数	2	A	8	F.	6	D
	↓	↓	↓	↓	↓	↓
二进制数	0010	1010	1000	1111.	0110	1101

所以，2A8F.6DH＝ 0010101010001111.01101101B。

1.2.3 计算机中的二进制数表示

在计算机中，用于表示数量大小的数据称为数值数据。讨论数值数据时常涉及两个

概念：表数范围和表数精度。表数范围是指一种类型的数据所能表示的最大值和最小值；对表数精度，通常用实数值能给出的有效数字的位数表示。在计算机中，表数范围和表数精度的大小与用多少个二进制位表示某类数据及怎样对某些位编码有关。

1. 定点小数的表示

定点小数是指小数点准确固定在数据某个位置上的小数。为方便起见，通常都把小数点固定在最高数据位的左边，称为纯小数。如果考虑数的符号，小数点的前边可以再设符号位。据此，任意一个小数都可写为

$$N = N_s \cdot N_{-1} N_{-2} \cdots N_{-(m-1)} N_{-m}$$

若用 $m+1$ 个二进制位表示上述小数，则可以用最高(最左)位表示该数的符号(假设用 0 表示正，用 1 表示负)，如上式中的 N_s，后边的 m 位表示小数的数值部分。由于规定了小数点放在数值部分的最左边，所以小数点不需明确表示出来。

定点小数的表数范围很小，对于用 $m+1$ 个二进制位表示的小数，其表数范围为

$$|N| \leqslant 1 - 2^{-m}$$

采用这种表示法，用户在算题时，需要先将参加运算的数通过一个合适的“比例因子”转化为绝对值小于 1 的纯小数，并保证运算的中间结果和最终结果的绝对值也都小于 1，在输出真正结果时，再按相应比例将结果扩大。

定点小数表示法主要用在早期计算机中，它比较节省硬件。随着硬件成本的大幅降低，现代通用计算机中都能够处理包括定点小数在内的多种类型的数值了。

2. 整数的表示

整数所表示的数据的最小单位为 1，可以认为它是小数点定在数据的最低位右边的一种数据。与定点小数类似，如果要考虑数的符号，整数的符号位也在最高位，任意一个带符号的整数都可表示为

$$N = N_s N_{n-1} \cdots N_1 N_0$$

式中：N_s 表示符号，后边的 n 位表示数值部分。对于这种用 $n+1$ 个二进制位表示的带符号的二进制数，其表数范围为

$$|N| \leqslant 2^n - 1$$

若不考虑数的符号，即所有的 $n+1$ 个二进制位都是有效数字，此时最高位 N_s 的权值为 2^n，则表数范围等于

$$0 \leqslant N \leqslant 2^{n+1} - 1$$

在计算机系统中，通常可用几种不同的二进制位数表示一个整数，如 8 位、16 位、32 位、64 位等，这些位数也称为字长。不同字长的整数所占用的存储器空间不同，其能够表达的数值的范围也不同(即上式中的 n 不同)。

3. 浮点数的表示

所谓浮点数，是指小数点的位置可以左右移动的数据，可用下式表示：

$$N = \pm R^E \times M$$

式中：M(Mantissa)：浮点数的尾数，或称有效数字，通常是纯小数；R(Radix)：阶码的基数，表示阶码采用的数制，计算机中一般规定 R 为 2、8 或 16，是一个常数，与尾数的基数相同，例如尾数为二进制，则 R 也为 2，同一种机器的 R 值是固定不变的，所以不需在浮点数中明确表示出来，而是隐含约定的，因此计算机中的浮点数只需表示出阶码和尾数部分；E(Exponent)：阶码，即是指数值，为带符号整数。

除此之外，浮点数的表示中还有 E_s 和 M_s 两个符号。

E_s：阶符，表示阶码的符号，即指数的符号，决定浮点数范围的大小。

M_s：尾符，尾数的符号位，安排在最高位。它也是整个浮点数的符号位，表示该浮点数的正负。

在计算机系统中，典型的浮点数格式如图 1-9 所示。

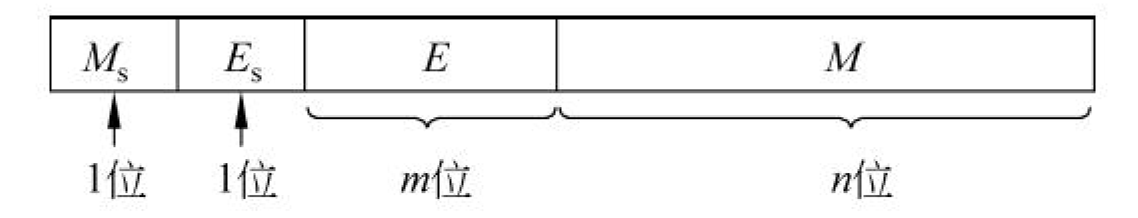


图 1-9　典型的浮点数格式

从浮点数的定义知，如果不作明确规定，同一个浮点数的表示将不是唯一的。例如，0.5 可以表示为 0.05×10^{1}，50×10^{-2} 等。为了便于浮点数之间的运算和比较，也为了提高数据的表示精度，规定计算机内浮点数的尾数部分用纯小数表示，即小数点右边第 1 位不为 0，称为规格化浮点数。对不满足要求的数，可通过修改阶码并同时左右移动小数点位置的方法使其变为规格化浮点数，这个过程也称为浮点数的规格化。

浮点数的表数范围主要由阶码决定，精度则主要由尾数决定。

1.2.4　二进制编码

计算机能够直接识别和处理的只有二进制数，但人们在生活、学习和工作中则更习惯于用十进制数，所以在某些情况下也希望计算机能直接处理十进制形式表示的数据。此外，现代计算机不仅要处理数值领域的问题，还需要处理大量非数值领域的问题，如文字处理、信息发布、数据库系统等，这就要求计算机还应能够识别和处理文字、字符和各种符号，如

数字——0、1、…、9；

字母——26 个大小写的英文字母：A、B、…、Z、a、b、…、z；

专用符号——+、-、*、/、↑、$、%、…；

控制字符——CR(回车)、LF(换行)、BEL(响铃)、…。

所有这些字符、符号以及十进制数最终都必须转换为二进制格式的代码才能为计算机所处理，即字符和十进制数都必须用若干位二进制码来表示，这就是信息和数据的二进制编码。

1. 二进制编码的十进制数

用二进制编码表示的十进制数，称为二-十进制(Binary Coded Decimal，BCD)码，它

的特点是保留了十进制的权，而数字则用 0 和 1 的组合编码来表示。用二进制码表示十进制数，至少需要的二进制位数为 $\log_2 10$，取整数等于 4，即至少需要 4 位二进制码才能表示 1 位十进制数。4 位二进制码有 16 种组合，而十进制数只有 10 个符号，选择哪 10 个符号来表示十进制的 0～9 有多种可行方案，下面只介绍最常用的一种 BCD 码，即 8421 码。

1）8421 码

8421 BCD 码（以下就简称“BCD 码”）用 4 位二进制编码表示 1 位十进制数，其4 位二进制编码的每一位都有特定的权值，从左至右分别为 $2^3=8$、$2^2=4$、$2^1=2$、$2^0=1$，故称其为 8421 码。

需要注意的是，BCD 码表示的是十进制数，只有 0～9 这 10 个有效数字，4 位二进制码的其余 6 种组合（1010～1111）是有效的十六进制数，但对 BCD 码是非法的。表 1-2 给出了 BCD 码与十进制数的对应关系。

表 1-2　BCD 码与十进制数的对应关系

十进制数	8421 码	十进制数	8421 码
0	0000	5	0101
1	0001	6	0110
2	0010	7	0111
3	0011	8	1000
4	0100	9	1001

BCD 码的计数规律与十进制数相同，即逢十进一。在书写上，每一个 4 位写在一起，以表示十进制的 1 位，结尾处加标记符 BCD，如 $(0011\ 0100)_{BCD}$ 表示十进制数 34。

2）BCD 码与十进制数、二进制数的转换

一个十进制数用 BCD 码来表示是非常简单的，只要对十进制数的每一位按表 1-2 的对应关系单独进行转换即可。

【例 1-10】 试把十进制数 234.15 写成 BCD 码的表示形式。

解： 将 234.15 的每一位用对应的 BCD 码表示，可得

$$(234.15)_{10}=(0010\ 0011\ 0100.0001\ 0101)_{BCD}$$

同样，也能够很容易地由 BCD 码得出其对应的十进制数。如 BCD 码 0110 0011 1001 1000.0101 0010 对应的十进制数为 6398.52。

BCD 码与二进制数之间的转换要稍微麻烦一些，一般需要先转换为十进制数。

【例 1-11】 将 BCD 码 $(0001\ 0001.0010\ 0101)_{BCD}$ 转换为二进制数。

解：

$$(0001\ 0001.0010\ 0101)_{BCD}=(11.25)_{10}$$

$$(11.25)_{10}=(1011.01)_2$$

所以，$(0001\ 0001.0010\ 0101)_{BCD}=(1011.01)_2$。

【例 1-12】 将二进制数 01000111 转换为 BCD 码。

解：

$$(01000111)_2=(71)_{10}=(0111\ 0001)_{BCD}$$

3）计算机中BCD码的存储方式

计算机的存储单元通常以字节（8个二进制位）为最小单元，很多操作也是以字节为单位进行的，在一个字节中如何存放BCD码有两种方式，即压缩的BCD码和非压缩的BCD码。

在一个字节中存放两个4位的BCD码，这种方式称为压缩BCD码表示法。在采用压缩BCD码表示十进制数时，一个字节就表示两位十进制数，例如10010010B表示十进制数92。

非压缩的BCD码（又称扩展BCD码）表示法是每个字节只存放一个BCD码，即低4位为有效BCD数，高4位全为0。例如同样是十进制数92，用非压缩BCD码就表示为00001001 00000010。

2. 字符的编码

各种字符和符号也必须按特定的规则用二进制编码才能在机器中表示。目前在微型计算机中普遍采用的字符编码系统是ASCII码（American Standard Code for Information Interchange，美国国家标准信息交换码），ASCII字符编码表参见书后的附录A。它用7位二进制编码来表示128个字符和符号。

微型计算机中一个字节为8位，一般规定一个ASCII码存放在字节的低7位，字节最高位D_7位恒为0。这样，用一个字节来表示一个ASCII字符编码，则数字0～9的ASCII码为30H～39H，26个英文大写字母A～Z的ASCII码为41H～5AH，而26个英文小写字母a～z的ASCII码为61H～7AH。

数据在计算机内形成、存取和传送的过程中可能产生错误。为尽量减少和避免这类错误，除提高软硬件系统的可靠性外，也常在数据的编码上想办法，即采用带有一定特征的编码方法，在硬件线路的配合下，能够发现错误、确定错误的性质和位置，甚至实现自动改正错误。数据校验码就是这样一种能发现错误并具有自动改错能力的编码方法。

在ASCII码的传送中，最常用到的校验码是一种开销小、能发现一位数据出错的奇偶校验码。带有奇偶校验的ASCII码将最高位（D_7位）用作奇偶校验位，以校验数据传送中是否有一位出现错误。

偶校验的含义是：包括校验位在内的8位二进制码中1的个数为偶数；而奇校验的含义是：包括校验位在内的8位二进制码中1的个数为奇数。例如，大写字母A的ASCII码为$(1000001)_2$，具有偶校验的A的ASCII码是$(01000001)_2$，具有奇校验的A的ASCII码是$(11000001)_2$。

1.3 无符号二进制数的算术运算和逻辑运算

二进制数在表示上可分为无符号数和有符号数两种。所谓无符号数，就是不考虑数的符号，数中的每一位0或1都是有效的或有意义的数据。

有符号数则不同于无符号数。在十进制数中,正数和负数分别用＋和－来表示,但计算机不能直接识别这两种符号。因此在计算机中,表示二进制数的符号仍然是用 0 和 1,即在需要考虑数据符号的有符号数中,一个数的最高位的 0 或 1 表示的是该数的性质,即是正数或负数,而不再是数据本身。有符号数的表示和运算在计算机中是非常重要的内容,将在 1.4 节中讨论。

1.3.1 二进制的算术运算

由于二进制数中只有 0 和 1 两个数,故其运算规则比十进制数要简单得多。

1. 加法运算

二进制的加法运算遵循如下法则。

0＋0＝0　0＋1＝1　1＋0＝1　1＋1＝0(有进位)

【例 1-13】 计算 10110110B＋01101100B＝(?)B

解:

进　位	111111000
被加数	10110110
加　数 ＋)	01101100
	100100010

即 10110110B＋01101100B＝100100010B。

2. 减法运算

二进制数的减法遵循如下法则。

0－0＝0　1－0＝1　1－1＝0　0－1＝1(有借位)

【例 1-14】 计算 11000100B－00100101B＝(?)B

解:

借　位	01111110
被减数	11000100
减　数 －)	00100101
	10011111

即 11000100B－00100101B＝10011111B。

3. 乘法运算

二进制数的乘法法则如下。

0×0＝0　0×1＝0　1×0＝0　1×1＝1

即,仅当两个 1 相乘时结果为 1,否则结果为 0。所以,二进制数的乘法是非常简单的。若乘数位为 1,就将被乘数照抄加于中间结果;若乘数位为 0,则加 0 于中间结果,只是在相加时要将每次中间结果的最后一位与相应的乘数位对齐。

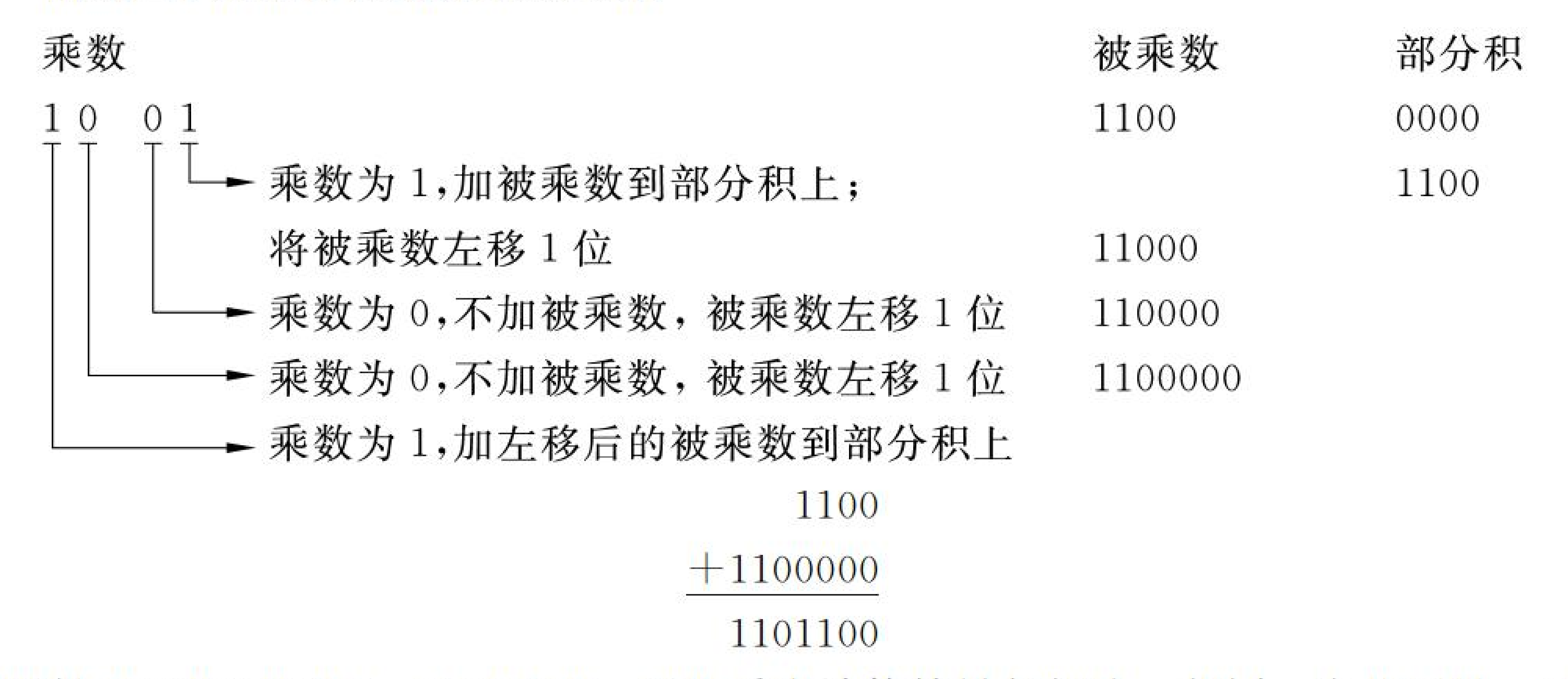
【**例 1-15**】 求两个二进制数 1100B 与 1001B 的乘积。

解法一：按照十进制的乘法过程有

```
      1100        被乘数
  ×   1001        乘  数
 ----------      --------
      1100        部分积
     0000
    0000
   1100
 ----------      --------
   1101100        乘  积
```

可得 1100B×1001B=1101100B。

解法二：采用移位加的方法，则有



即可得 1100B×1001B=1101100B。可以看出计算结果与解法一相同。由此可见，二进制的乘法运算可以转换为加法和移位的运算。事实上，在计算机中乘法运算就是这样做的。每左移一位，相当于乘以 2，而左移 n 位就相当于乘以 2^n。

4. 除法运算

除法是乘法的逆运算。所以二进制数的除法运算也可转换为减法和右移运算。每右移一位相当于除以 2，右移 n 位就相当于除以 2^n。

1.3.2 无符号数的表示范围

1. 无符号二进制数的表示范围

一个 n 位无符号二进制数 X，其可表示数的范围为

$$0 \leqslant X \leqslant 2^n - 1$$

例如一个 8 位的二进制数，即 $n=8$，其表示范围为 $0 \sim 2^8-1$，即 00H～FFH（0～255）。若运算结果超出数的可表示范围，则会产生溢出，得到不正确的结果。

【**例 1-16**】 计算 10110111B+01001101B=(?)B

解：

$$\begin{array}{r} 10110111 \\ +01001101 \\ \hline 1\quad 00000100 \end{array}$$

由例 1-16 的结果可得，上面两个 8 位二进制数相加的结果为 9 位，超出了 8 位数的可表示范围。若仅取 8 位字长（00000100B），结果显然错误，这种情况称为溢出。事实上，$(10110111)_2=(183)_{10}$，$(01001101)_2=(77)_{10}$，则 183＋77＝260，大于 8 位二进制数所能表示的最大值 255，所以最高位的进位（代表了 256）就给丢失了，这样最后的结果变成了 260－256＝4，即 00000100B。

2. 无符号二进制数的溢出判断

对两个无符号二进制数的加减运算，若最高有效位 D_i 向更高位有进位（或相减有借位），则产生溢出。例如在例 1-16 中，两个 8 位无符号二进制数相加，最高有效位（即 D_7 位）向更高位（即 D_8 位）有进位，结果就出现了溢出。

对乘法运算，由于两个 8 位数相乘，乘积为 16 位；两个 16 位数相乘，乘积为 32 位。故乘法运算无溢出问题。对除法运算，当除数过小时会产生溢出，此时将使系统产生一次溢出中断。有关中断的理论将在第 6 章详细介绍。

1.3.3 二进制数的逻辑运算

算术运算是将一个数据作为一个整体来考虑的，而逻辑运算则是对数据的每一位按位进行操作，这意味着逻辑运算没有进位和借位。基本逻辑运算包括“与”、“或”、“非”、“异或”4 种运算。

1. “与”运算

“与”运算的操作是实现两个数按位相“与”，用符号∧表示。其规则为

$$1\wedge 1=1\quad 1\wedge 0=0\quad 0\wedge 1=0\quad 0\wedge 0=0$$

即参加“与”操作的两位中只要有一位为 0，则“与”的结果就为 0，仅当两位均为 1 时，其结果才为 1。

试比较二进制数的“与”运算规则和乘法运算规则，思考一下二者之间的相同之处和不同之处。

【例 1-17】 计算 10110110B∧10010011B＝(?)B

解：

$$\begin{array}{r} 10110110 \\ \wedge 10010011 \\ \hline 10010010 \end{array}$$

即 10110110B∧10010011B＝10010010B。

2. “或”运算

“或”运算的操作是实现两个数按位相“或”，用符号∨表示。其规则为

$$0 \vee 0 = 0 \quad 0 \vee 1 = 1 \quad 1 \vee 0 = 1 \quad 1 \vee 1 = 1$$

即参加“或”操作的两位中只要有一位为 1，则“或”的结果就为 1，仅当两位均为 0 时，其结果才为 0。

> 试比较二进制数的“或”运算规则和加法运算规则，思考一下二者之间的相同之处和不同之处。

【例 1-18】 计算 11011001B∨10010110B＝(?)B

解：

$$\begin{array}{r} 11011001 \\ \vee\, 10010110 \\ \hline 11011111 \end{array}$$

即 11011001B∨10010110B＝11011111B。

3. “非”运算

“非”运算的操作为将一个数的每一位按位取反，即 1 的“非”为 0，而 0 的“非”为 1。其运算符为“—”，例如：

$$\overline{1} = 0 \quad \overline{0} = 1$$

【例 1-19】 求数 11011001 的非。

解：只要对 11011001 按位取反即可。

$$\overline{11011001B} = 00100110B$$

4. “异或”运算

“异或”运算的操作是实现两个数按位相异或。两位相同，则结果为 0；两位相异，则结果为 1。“异或”运算符用符号⊕表示。

$$0 \oplus 0 = 0 \quad 1 \oplus 1 = 0 \quad 0 \oplus 1 = 1 \quad 1 \oplus 0 = 1$$

【例 1-20】 计算 11010011B⊕10100110B＝(?)B

解：

$$\begin{array}{r} 11010011 \\ \oplus\, 10100110 \\ \hline 01110101 \end{array}$$

即 11010011B⊕10100110B＝01110101B。

> 试比较二进制数的“异或”运算规则和减法运算规则，思考一下二者之间的异同。

1.3.4 基本逻辑门及常用逻辑部件

本节介绍几种后面要用到的、最常用的计算机基本逻辑部件，已经学过数字电路的读者，可跳过本节。对这些逻辑部件，我们也仅是从应用的角度出发，只关心它们的逻辑功能和外部引线连接，而不关心其内部的电路构成。

1. 与门

与门(AND gate)是对多个逻辑变量进行“与”运算的门电路。若输入的逻辑变量为 A 和 B，则通过与门输出的结果 Y 可表示为

$$Y = A \wedge B$$

表 1-3 给出了与门的真值表。当输入 A 和 B 均为 1 时，输出 Y 才为 1；A 和 B 中只要有一个为 0，则 Y 就等于 0。从电路的角度来说，若采用正逻辑，则仅当与门的输入 A 和 B 都是高电平时，输出 Y 才是高电平，否则 Y 就输出低电平。

在电路连接上，与门常用图 1-10 所示的逻辑符号表示。

表 1-3 与门的真值表

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

(a) IEEE推荐的符号　　(b) 中国国家标准规定使用的符号

图 1-10 与门的逻辑符号

需要说明的是：

- 图 1-10 中仅画出了 2 位输入(A 和 B)，实际的与门电路可以有多位输入(以下“或门”类同)。
- 图 1-10 给出了与门的两种表示方法。其中，图 1-10(a)为 IEEE 推荐符号，图 1-10(b)为中国国家标准规定使用的符号。这两种图符目前均可以使用(以下类同)。为描述方便，本书后续内容的描述以中国国家标准规定的图符为主。

2. 或门

或门(OR gate)是对多个逻辑变量进行“或”运算的门电路。若输入的逻辑变量为 A 和 B，则通过或门输出的结果 Y 可表示为

$$Y = A \vee B$$

即两个输入变量 A 和 B 中任意一个为 1，输出 Y 就为 1；仅当 A 和 B 都为 0 时 Y 才为 0。从电路的角度来说，当或门的输入 A 和 B 只要有一个是高电平，输出 Y 就为高电平，否则 Y 就输出低电平。

或门的逻辑符号如图 1-11 所示，其真值表如表 1-4 所示。

表 1-4　或门的真值表

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1



(a) IEEE推荐的符号　　(b) 中国国家标准规定使用的符号

图 1-11　或门的逻辑符号

3. 非门

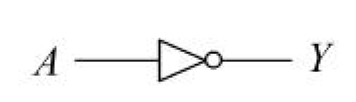
非门(NOT gate)又称为反相器,是对单一逻辑变量进行“非”运算的门电路。其输入变量 A 与输出变量 Y 之间的关系可用下式表示

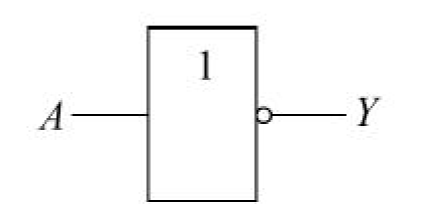
$$Y = \overline{A}$$

非运算也称求反运算,变量 A 上的上划线“—”在数字电路中表示反相之意。非门的逻辑符号如图 1-12 所示,其真值表如表 1-5 所示。

表 1-5　非门的真值表

A	Y
0	1
1	0





(a) IEEE推荐的符号　　(b) 中国国家标准规定使用的符号

图 1-12　非门的逻辑符号

4. 与非门

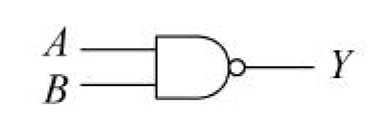
与非门(NAND gate)是“与”门与“非”门的结合。若输入变量为 A 和 B,则先对输入 A 和 B 进行“与”运算,再对结果进行“非”运算。运算表达式为

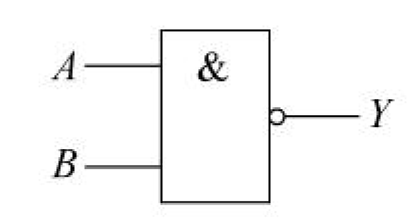
$$Y = \overline{A \wedge B}$$

与非门的逻辑符号如图 1-13 所示,逻辑符号图中的小圆圈表示“非”(本书将始终采用这种表示方法)。与非门的真值表如表 1-6 所示。

表 1-6　与非门的真值表

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0





(a) IEEE推荐的符号　　(b) 中国国家标准规定使用的符号

图 1-13　与非门的逻辑符号

5. 或非门

和与非门类似,或非门(NOR gate)是“或”门与“非”门的结合,即先对输入 A 和 B 进行“或”运算,再对其结果进行“非”运算。其运算表达式为

$$Y = \overline{A \vee B}$$

或非门的逻辑符号如图 1-14 所示，真值表如表 1-7 所示。

表 1-7　或非门的真值表

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

(a) IEEE推荐的符号　　(b) 中国国家标准规定使用的符号

图 1-14　或非门的逻辑符号

6. 译码器

在计算机系统中，常常需要将不同的地址信号通过一定的控制电路转换为对某一芯片的片选信号，这个控制电路称为译码电路，它所对应的逻辑部件就称为译码器。也可以说，译码器的作用就是将一组输入信号转换为在某一时刻有一个确定的输出信号。

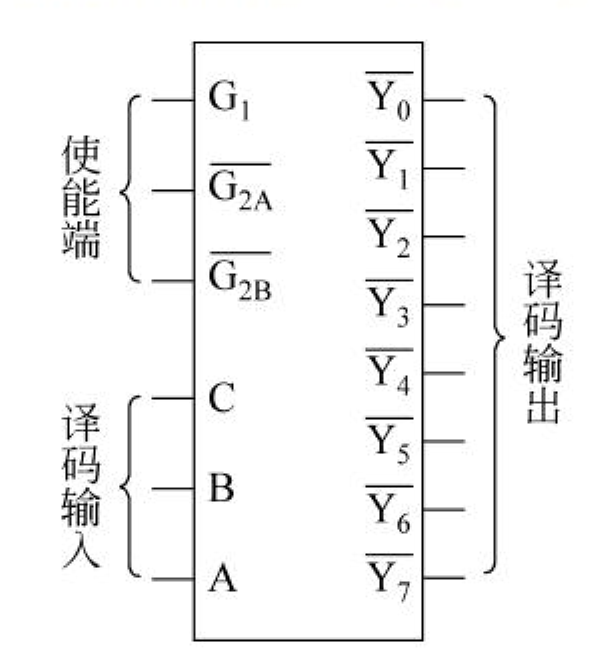


图 1-15　74LS138 的引脚功能图

译码器的种类很多，这里仅介绍一种常用的 3-8 线译码器 74LS138。74LS138 的引脚如图 1-15 所示。图中 G_1、G_{2A}、G_{2B}为译码器的 3 个使能输入端，它们共同决定了译码器当前是否被允许工作：当 $G_1=1$，$G_{2A}=G_{2B}=0$ 时，译码器处于使能状态(Enable)，否则就被禁止(Disable)。C、B、A 为译码器的 3 条输入线(输入的 3 位二进制代码分别代表了 8 种不同的状态)，它们的不同的状态组合决定了 8 个输出端 $Y_0\sim Y_7$ 的状态。74LS138 的功能表(也叫真值表)如表 1-8 所示，表中电平为正逻辑，即高电平表示逻辑 1，低电平表示逻辑 0，×表示不定，# 表示该信号低电平有效(与上横线标注¯含义相同)。

表 1-8　74LS138 功能表

使　能　端			输入端			输　出　端							
G_1	#G_{2A}	#G_{2B}	C	B	A	#Y_0	#Y_1	#Y_2	#Y_3	#Y_4	#Y_5	#Y_6	#Y_7
×	1	1	×	×	×	1	1	1	1	1	1	1	1
0	×	×	×	×	×	1	1	1	1	1	1	1	1
1	**0**	**0**	**0**	**0**	**0**	**0**	1	1	1	1	1	1	1
1	**0**	**0**	**0**	**0**	**1**	1	**0**	1	1	1	1	1	1
1	**0**	**0**	**0**	**1**	**0**	1	1	**0**	1	1	1	1	1
1	**0**	**0**	**0**	**1**	**1**	1	1	1	**0**	1	1	1	1
1	**0**	**0**	**1**	**0**	**0**	1	1	1	1	**0**	1	1	1
1	**0**	**0**	**1**	**0**	**1**	1	1	1	1	1	**0**	1	1
1	**0**	**0**	**1**	**1**	**0**	1	1	1	1	1	1	**0**	1
1	**0**	**0**	**1**	**1**	**1**	1	1	1	1	1	1	1	**0**

1.4 有符号二进制数的表示及运算

前面讨论了不涉及数据符号的无符号数。但在数值运算中，常常需要考虑数值数据的符号数。由于计算机硬件系统不能直接识别＋和－这样的符号。所以，计算机中的符号数是由 0 和 1 来表示正负的。规定一个符号数的最高位为符号位，0 表示正，1 表示负。以 8 位字长为例，D_7 位就是符号位，D_6～D_0 为数值位，若字长为 16 位，则 D_{15} 为符号位，D_{14}～D_0 为数值位。这样，有符号数中的有效数值就比相同字长的无符号数要小了，因为其最高位代表符号，而不再是有效的数据。

【例 1-21】 ＋0010101B 在计算机中可表示为 00010101B，即十进制数的＋21；
－0010101B 在计算机中可表示为 10010101B，即十进制数的－21。

人们将符号数值化了的数称为机器数，如 00010101 和 10010101 就是机器数，而将原来的数值称为机器数的真值，如＋0010101 和－0010101。下面来讨论有符号机器数的表示方法及它们的运算规则。

1.4.1 有符号数的表示方法

计算机中的符号数有 3 种表示方法，即原码、反码和补码。它们均由符号位和数值部分组成，符号位的表示方法相同，都是用 1 表示“负”，用 0 表示“正”。

1. 原码

真值 X 的原码记为$[X]_原$。在原码表示法中，不论数的正负，数值部分均保持原真值不变。

【例 1-22】 已知真值 $X=+42$，$Y=-42$，求$[X]_原$和$[Y]_原$。

解：因为$(+42)_{10}=+0101010B$，$(-42)_{10}=-0101010B$，根据原码表示法，有

$$[X]_原=\underset{\uparrow\atop 符号位}{\underline{0}}\quad\underset{\uparrow\atop 数值部分}{\overline{0101010}}\qquad\qquad [X]_原=\underset{\uparrow\atop 符号位}{\underline{1}}\quad\underset{\uparrow\atop 数值部分}{\overline{0101010}}$$

原码的性质如下。

(1) 在原码表示法中，机器数的最高位是符号位，0 表示正号，1 表示负号，其余部分是数的绝对值，即$[X]_原$＝符号位＋$|X|$。

(2) 原码表示中的 0 有两种不同的表示形式，即＋0 和－0。

$$[+0]_原=00000000$$
$$[-0]_原=10000000$$

(3) 原码表示法的优点是简单、易于理解，与真值间的转换较为方便，用原码实现乘除运算的规则比较简单；缺点是进行加减运算时比较麻烦，要比较进行加减运算的两个数的符号、两个数的绝对值的大小，还要确定运算结果的正确的符号等。

若二进制数 $X=X_{n-1}X_{n-2}\cdots X_1X_0$，则原码表示的严格定义是

$$[X]_{原} = \begin{cases} X & 2^{n-1} > X \geqslant 0 \\ 2^{n-1} - X = 2^{n-1} + |X| & 0 \geqslant X > -2^{n-1} \end{cases} \quad (1.5)$$

2. 反码

真值 X 的反码记为$[X]_{反}$。对正数来讲，其表示方法同原码，即数值部分与真值相同；对负数来讲，其反码的数值部分为真值的各位按位取反，或者说负数的反码等于其对应正数的原码按位取反。例如，$[+127]_{反}=01111111$，$[-127]_{反}=10000000$。

【例 1-23】 已知真值 $X=+42$，$Y=-42$，求$[X]_{反}$和$[Y]_{反}$。

解：因为$(+42)_{10}=+0101010B$，$(-42)_{10}=-0101010B$，根据反码表示法，有

$$[X]_{反} = 00101010 \quad [Y]_{反} = 11010101$$

反码的性质如下。

(1) 在反码表示法中，机器数的最高位是符号位，0 表示正号，1 表示负号。

(2) 同原码一样，数 0 也有两种表示形式。

$$[+0]_{反} = 00000000$$
$$[-0]_{反} = 11111111$$

(3) 反码运算很不方便，数值 0 的表示也不唯一。目前在微处理器中已很少使用。

若二进制数 $X=X_{n-1}X_{n-2}\cdots X_1X_0$，则反码表示的严格定义是

$$[X]_{反} = \begin{cases} X & 2^{n-1} > X \geqslant 0 \\ (2^n - 1) + X & 0 \geqslant X > -2^{n-1} \end{cases} \quad (1.6)$$

3. 补码

真值 X 的补码记为$[X]_{补}$。补码是根据同余的概念得出的。由同余的概念可知，对一个数 X 有

$$X + nK = X(\text{mod } K) \quad (1.7)$$

式中：K 为模数；n 为任意整数。在模的意义下，数 X 就等于其本身加上它的模的任意整数倍之和。若设 n 为 1，$K=2^n$，则有

$$X = X + 2^n \quad (\text{mod } 2^n)$$

即

$$X = \begin{cases} X & 2^{n-1} > X \geqslant 0 \\ 2^n + X = 2^n - |X| & 0 > X \geqslant -2^{n-1} \end{cases} \quad (\text{mod } 2^n) \quad (1.8)$$

实际上，式(1.8)就是补码表示的定义。如设机器字长 $n=8$，则

$$[+1]_{补} = 00000001, \quad [-1]_{补} = 2^8 - |-1| = 11111111$$
$$[+127]_{补} = 01111111, \quad [-127]_{补} = 2^8 - |-127| = 10000001$$

补码的性质如下。

(1) 与原码和反码表示法相同，机器数的最高位是符号位，0 表示正号，1 表示负号。

(2) 正数的补码与它的原码和反码相同，即当 $X\geqslant 0$ 时，$[X]_{补}=[X]_{反}=[X]_{原}$。而负数的补码等于其符号位不变，数值部分的各位按位取反再加 1，即当 $X<0$ 时，$[X]_{补}=[X]_{反}+1$(也可以说，负数的补码等于其对应正数的补码包括符号位一起按位取反再加

1)。如：$[-127]_{补}=\overline{[+127]_{补}}+1=\overline{01111111}+1=10000001$。

(3) 数 0 的补码表示是唯一的。这点可由补码的定义得出。

$$[+0]_{补}=[+0]_{反}=[+0]_{原}=00000000$$

$$[-0]_{补}=[-0]_{反}+1=11111111+1=00000000(\bmod 2^8)$$

即对 8 位字长来讲，最高位的进位(2^8)按模 256 运算被舍掉，所以$[+0]_{补}=[-0]_{补}=00000000$。

(4) 对 8 位二进制数 10000000(16 位二进制数为 1000000000000000，依此类推)，在补码中它定义为−128(16 位二进制数 1000000000000000 定义为−32768)，而在原码中它表示−0，在反码中表示−127。

【例 1-24】 已知真值 $X=+0110100$，$Y=-0110100$，求$[X]_{补}$和$[Y]_{补}$。

解：这里 $X>0$，所以有

$$[X]_{补}=00110100$$

而 $Y<0$，所以有

$$[Y]_{补}=[Y]_{反}+1=11001011+1=11001100$$

1.4.2 补码数与十进制数之间的转换

要把一个用补码表示的二进制数转换为带符号的十进制数，首先应求出它的真值，然后再进行二-十进制转换即可。

1. 正数补码的转换

由于正数的补码就等于它的原码，即真值就是它的数值部分，也就是说，除符号位之外的其余数值位就是该数的真值。

【例 1-25】 已知$[X]_{补}=00101110$，求 X 的真值。

解：因为补码 00101110 的符号位为 0，是一个正数，它的数值部分就是它的真值，即

$$X=+0101110=(+46)_{10}$$

2. 负数补码的转换

负数的补码与其对应的正数补码之间存在如下关系：

$$[X]_{补}\xrightarrow{\text{按位取反加 1}}[-X]_{补}\xrightarrow{\text{按位取反加 1}}[X]_{补}$$

例如，若设 $X=+1$，则有 $-X=-1$；那么，$[X]_{补}=[+1]_{补}=00000001$，对其按位取反加 1，有$\overline{00000001}+1=11111111=[-1]_{补}$；反之，对$[-1]_{补}$按位取反也有$\overline{[-1]_{补}}+1=\overline{11111111}+1=00000001=[+1]_{补}$。

由此可得，当 X 为正数时，对其补码按位取反，结果是 $-X$ 的补码；当 X 为负数时，对其补码按位取反，结果就是 $+X$ 的补码。

所以，对负数补码再求补的结果就是该负数的绝对值。这样，负数补码转换为真值的方法就是：将此负数的补码数再求一次补(即将该负数补码的数值部分按位取反加 1)，所

得结果即是它的真值。

【例 1-26】 已知$[X]_补$=11010010,求 X 的真值。

解:因为补码 11010010 的符号位为"1",可知它是一个负数。要求得其真值需再对其取补码,即

$$X = [[X]_补]_补 = [11010010]_补 = -0101110 = (-46)_{10}$$

为什么要引进补码的概念呢?这是因为在计算机中,对于二进制的算术运算可以将乘法运算转换为加法和左移运算,而除法则可转换为减法和右移运算,故加、减、乘、除运算最终可归结为加、减和移位 3 种操作来完成。但在计算机中为了节省设备,一般只设置加法器而无减法器,这就需要将减法运算转化为加法运算,从而使在计算机中的二进制四则运算最终变成加法和移位两种操作。引进补码运算就是用来解决将减法运算转化为加法运算的。

1.4.3 补码的运算

补码运算有如下规则。

(1) 补码的加法规则:$[X+Y]_补=[X]_补+[Y]_补$。

(2) 补码的减法规则:$[X-Y]_补=[X]_补-[Y]_补=[X]_补+[-Y]_补$。

这里,$[-Y]_补$称为对补码数$[Y]_补$求变补。变补的规则为:对$[Y]_补$的每一位(包括符号位)按位取反加 1,则结果就是$[-Y]_补$。当然,也可以直接对$-Y$ 求补码,结果是一样的。

【例 1-27】 设 $X=+66$,$Y=-51$,求$[X+Y]_补=?$

解:由补码的加法运算规则知$[X+Y]_补=[X]_补+[Y]_补$。

先分别求出 X 和 Y 的补码:

$$X = (+66)_{10} = (+1000010)_2,\quad [X]_补 = 01000010$$

$$Y = (-51)_{10} = (-0110011)_2,\quad [Y]_补 = 11001101$$

再求$[X]_补+[Y]_补$得

```
    01000010
   +11001101
  ----------
  1 00001111
  ↑
自然丢失
```

所以,$[X+Y]_补=00001111=(+15)_{10}$。

在字长为 8 位的机器中,从第 7 位向上的进位是自然丢失的,故本例中做加法运算的结果与用补码做减法运算的结果相同,都是十进制数 15。

【例 1-28】 设 $X=+51$,$Y=+66$,求$[X-Y]_补=?$

解:因为

$$[X-Y]_补 = [X]_补 + [-Y]_补$$

$$X = (+51)_{10} = (+0110011)_2,\quad [X]_补 = 00110011$$

$$-Y = (-66)_{10} = (-1000010)_2,\quad [-Y]_补 = 10111110$$

求$[X]_补+[-Y]_补$得

$$
\begin{array}{r}
00110011 \\
+10111110 \\
\hline
11110001
\end{array}
$$

所以，$[X-Y]_{补}=11110001$。

由补码运算规则知，两补码相加的结果为和的补码，现在和的符号位为 1，表示和为负数。按照负数补码转换真值的原则，其符号位用“－”表示，数值部分按位取反加 1，得出真值：－0001111。故通过补码相加后，和为十进制数－15。

由此说明，当两个带符号数用补码表示时，减法运算可转换为加法运算。

还可通过钟表来说明补码的概念。假如有一只钟表的时针指在 9 点，若要拨到 4 点，有两种拨法：

逆时针拨，倒拨 5 小时 9－5＝4

顺时针拨，正拨 7 小时 9＋7＝12＋4＝4(mod 12)

此处的 12 就是时钟系统中的模(计数系统最大的数)，它是自然丢失的，故顺时针拨 7 个字相当于逆时针拨 5 个字，结果都是 4。

对模 12 而言，9－5＝9＋7，这时就称 7 为－5 的以 12 为模的补数，即

$$[-5]_{补}=12-5=7$$

这与上边的表达式是一致的。这样就有

$$9-5=9+(-5)=9+(12-5)=9+7=\underbrace{12}_{\text{模自然丢失}}+4=4$$

在二进制数系统中，模为 2^n（n 为字长）。若字长为 8 位，则模为 $2^8=(256)_{10}$。

当一个负数用补码表示时，就可以将减法运算转换为加法运算。如在例 1-27 中，(66－51)可写成：66－51＝66＋(－51)＝66＋(256－51)＝66＋205＝256＋15＝15(mod 256)。

可见在模为 2^8 的情况下，(66－51)与(66＋205)的结果是相同的。也就是说，对模为 256 来说，－51 与 205 互为补数，这里－51 的补码二进制数为 11001101，即是十进制数的 205(把 11001101 看成无符号数时为 205，若看成有符号数为－51)人们正是利用了负数的补码概念，把减法运算转换为加法运算。但要注意，这里负数($-X$)的补码是利用 2^8-X 来得到的，仍没有避免减法运算，实际上，根据负数补码的定义 $[X]_{补}=[X]_{反}+1$，就可避免求补过程中的减法运算，使补码运算具有实用价值。

> 在微机中，凡是有符号数都一定是用补码表示的，所以运算的结果也是用补码表示的。

1.4.4 有符号数的表示范围

1. 有符号数的表示范围

(1) 对 8 位二进制数，原码、反码和补码所能表示的范围如下。

① 原码：11111111B～01111111B(－127～＋127)。

② 反码：10000000B～01111111B(－127～＋127)。

③ 补码：10000000B～01111111B(－128～＋127)。

(2) 对16位二进制数,原码、反码和补码所能表示的范围如下。

① 原码：FFFFH～7FFFH(－32767～＋32767)。

② 反码：8000H～7FFFH(－32767～＋32767)。

③ 补码：8000H～7FFFH(－32768～＋32767)。

2. 有符号数运算时的溢出判断

在两个有符号数进行加减运算时,如果运算结果超出上述可表示的有效范围,就会发生溢出,使计算结果出错。显然,溢出只能出现在两个同符号数相加或两个异符号数相减的情况下。判断有符号数运算是否溢出,有下述规则。

在两个同符号数相加或异符号数相减时：

① 如果次高位向最高位有进位(或借位),而最高位向上无进位(或借位),则结果发生溢出；

② 反过来,如果次高位向最高位无进位(或借位),而最高位向上有进位(或借位),则结果也发生溢出。

对于8位二进制数,若D_6位产生的进位(或借位)记为C_6,D_7位产生的进位(或借位)记为C_7,那么上述两种情况也可表述为：

在两个带符号二进制数相加或相减时,若$C_7 \oplus C_6 = 1$,则结果产生溢出。

【例 1-29】 用二进制补码计算(＋72)＋(＋98)＝(?)

解：

$$(+72)_{10}=(+1001000)_2,(+1001000)_{补}=01001000$$

$$(+98)_{10}=(+1100010)_2,(+1100010)_{补}=01100010$$

01001000B	＋72
＋01100010B	＋98
10101010B	－86

例1-29中,两个正数相加,结果(补码)变成了负值,显然是错误的。原因是由于(＋72)＋(＋98)＝＋170＞＋127,超出了8位二进制补码的表示范围,结果产生溢出,导致出错。在计算中,从$C_6=1$,$C_7=0$就可判断出结果溢出。

【例 1-30】 用二进制补码计算(－83)＋(－80)＝(?)

解：

$$(-83)_{10}=(-1010011)_2,(-1010011)_{补}=10101101$$

$$(-80)_{10}=(-1010000)_2,(+1010000)_{补}=10110000$$

10101101B	－83
＋10110000B	－80
进位自然丢失→ [1] 01011101B	＋93

例1-30中,两个负数相加,结果变成了正值。原因就是(－83)＋(－80)＝－163＜

-128,超出了 8 位二进制补码的表示范围,使结果产生了溢出(由 $C_6=0$,$C_7=1$ 就可直接判断)。

以上是两个同符号数相加,当结果超出二进制补码的表示范围时将产生溢出。而对两个异符号数相减,同样有可能产生溢出,使结果出错。

【例 1-31】 用二进制补码计算(+72)-(-98)=(?)

解:

$$(+72)_{10}=(+1001000)_2,(+1001000)_补=01001000$$

$$(-(-98)_{10})=(+1100010)_2,(+1100010)_补=01100010$$

$$\begin{array}{rr} 01001000B & +72 \\ \underline{+01100010B} & \underline{+98} \\ 10101010B & -86 \end{array}$$

由计算过程得:$C_6=1$,$C_7=0$,知结果产生溢出。

由例 1-31 的讨论可知,无符号数与有符号数产生溢出的条件因各自可表示数的范围不同而不同。无符号数的溢出判断仅看最高位向上是否有进(借)位;而有符号数有无溢出产生,需要看次高位与最高位两位的进(借)位情况。两位都产生进(借)位或都没有产生进(借)位,则结果无溢出;否则结果产生溢出。运算时产生溢出,其结果肯定不正确。计算机对溢出的处理,一般是产生一个自陷中断,通知用户采取某种措施。

习　　题

1.1 计算机中常用的记数制有哪些?

1.2 请说明机器数和真值的区别。

1.3 完成下列数制的转换。

(1) 10100110B=(　　)D=(　　)H。

(2) 0.11B=(　　)D。

(3) 253.25=(　　)B=(　　)H。

(4) 1011011.101B=(　　)H=(　　)BCD。

1.4 8 位和 16 位二进制数的原码、补码和反码可表示的数的范围分别是多少?

1.5 写出下列真值对应的原码和补码的形式。

(1) $X=-1110011B$。

(2) $X=-71D$。

(3) $X=+1001001B$。

1.6 写出符号数 10110101B 的反码和补码。

1.7 已知 X 和 Y 的真值,求$[X+Y]_补=$?

(1) $X=-1110111B$,$Y=+1011010B$。

(2) $X=56$,$Y=-21$。

1.8 已知 $X=-1101001B$,$Y=-1010110B$,用补码方法求 $X-Y=$?

1.9　若给字符 4 和 9 的 ASCII 码加奇校验，应是多少？若加偶校验呢？

1.10　若与门的输入端 A、B、C 的状态分别为 1、0、1，则该与门的输出端是什么状态？若将这 3 位信号连接到或门，那么或门的输出又是什么状态？

1.11　要使与非门输出 0，则与非门输入端各位的状态应该是(　　)；如果使与非门输出 1，其输入端各位的状态又是什么？

1.12　如果 74LS138 译码器的 C、B、A 这 3 个输入端的状态为 011，此时该译码器的 8 个输出端中哪一个会输出 0？

1.13　图 1-16 中，Y_1＝？ Y_2＝？ Y_3＝？ 138 译码器哪一个输出端会输出低电平？

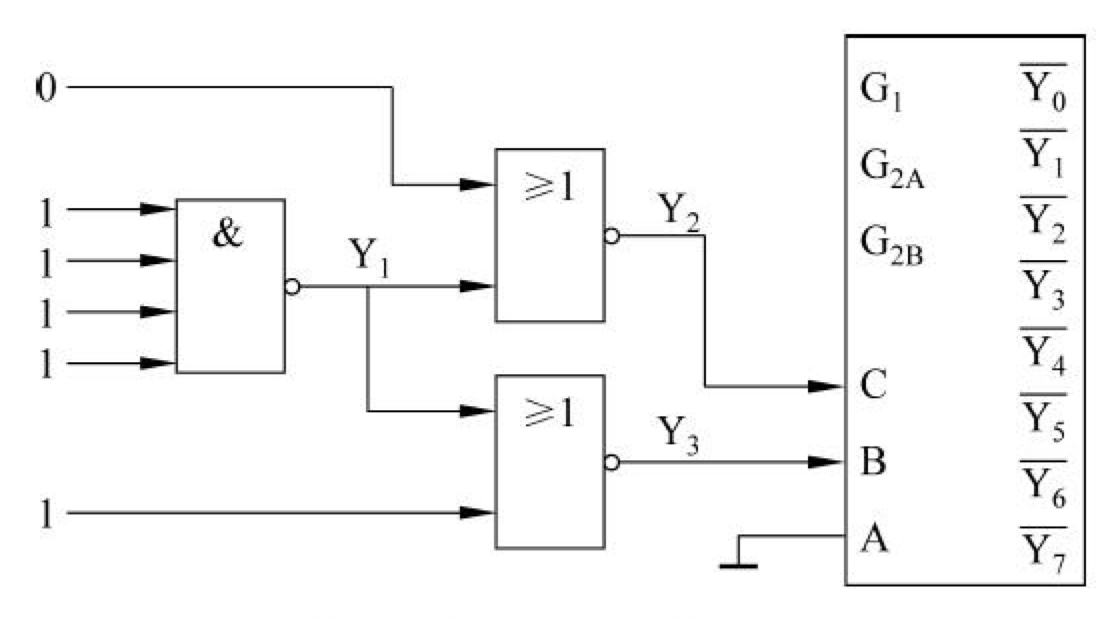


图 1-16　138 译码电路



第2章 微处理器与总线

引言：

无论是利用单片机技术还是利用微机系统，家庭安全防盗系统的控制中心都是微处理器。如今在世界范围内，Intel公司生产的CPU都是市场上的主流产品。其产品分为Celeron（赛扬）、Pentium（奔腾）、Core（酷睿）、Core i（酷睿 i）等四大系列，目前都是双核或多核产品，其中的Core和Core i系列更是多核技术出现后的产物。本书第1章中已提到多核的概念。所谓多核，是指在一块处理器中集成了多个功能相同的计算内核。它们虽然在架构上与单核处理器有较大不同，但核心的基本工作原理是类似的。本着本科学习以基本原理为主的原则，本章仅以Intel 80x86系列微处理器中的三种典型CPU（8088、80386和Pentium 4）为例，介绍微处理器的结构及其工作原理。对多核技术，仅简要介绍其基本概念，以及多核和多处理器技术之间的区别。

虽然总线是计算机硬件系统的一个重要部件，但限于篇幅，本章仅简要介绍总线的一般概念、主要功能及常用的总线接口标准。

通过本章的学习，读者将对微型机硬件系统中两大部件的基本构成及工作原理有一定的了解。

教学目的：

(1) 了解微处理器的一般结构和功能；
(2) 理解8088 CPU的外部引线及主要引线功能；
(3) 深入理解8088 CPU的结构特点、内部寄存器功能及工作时序；
(4) 理解80386和Pentium 4 CPU的3种工作模式；
(5) 了解80386和Pentium 4 CPU的结构特点；
(6) 了解流水线技术的一般概念；
(7) 理解总线的一般概念、分类方法及主要功能；
(8) 了解现代微机系统的总线结构；
(9) 了解常用的系统总线和外设总线标准。
(10) 了解多核技术的一般概念。

2.1 微处理器概述

微处理器(CPU)是计算机系统的核心部件，控制和协调着整个计算机系统的工作，

主要具有以下几项基本功能。

(1) 能够进行算术运算和逻辑运算。

(2) 能对指令进行译码、寄存并执行指令所规定的操作。

(3) 具有与存储器和 I/O 接口进行数据通信的能力。

(4) 少量数据的暂存。

(5) 能够提供这个系统所需的定时和控制信号。

(6) 能够响应输入输出设备发出的中断请求。

评价 CPU 性能的指标很多,包括工作频率、指令系统功能、内部缓存容量以及字长等,这里仅说一下字长。所谓字长,是指 CPU 在单位时间内(同一时间)能够一次处理的二进制数的位数,通常是 CPU 内部寄存器的位数及内部数据总线的位数。人们常说 16 位机、32 位机,其实是表示该计算机中微处理器可同时操作的二进制码的位数。对微型机来讲,有 8 位、16 位、32 位 CPU 等,其含义是同时可操作 8 位、16 位或 32 位二进制码。目前的主流 CPU 都是 64 位的,即一次可处理 64 位二进制数。

微处理器内部总体上由 3 个部分组成,即运算器、控制器和寄存器组,寄存器组又可视为运算器部件中的一部分。下面分别来看一下它们的组成及功能。

2.1.1 运算器

运算器由算术逻辑单元(Arithmetic Logical Unit,ALU)、通用或专用寄存器组及内部总线 3 个部分组成,其核心功能是实现数据的算术运算和逻辑运算,所以有时也将运算器称为算术逻辑运算单元。

ALU 的内部包括负责加、减、乘、除运算的加法器以及实现与、或、非、异或等逻辑运算的逻辑运算功能部件。一位算术逻辑运算单元的结构示意图如图 2-1 所示。

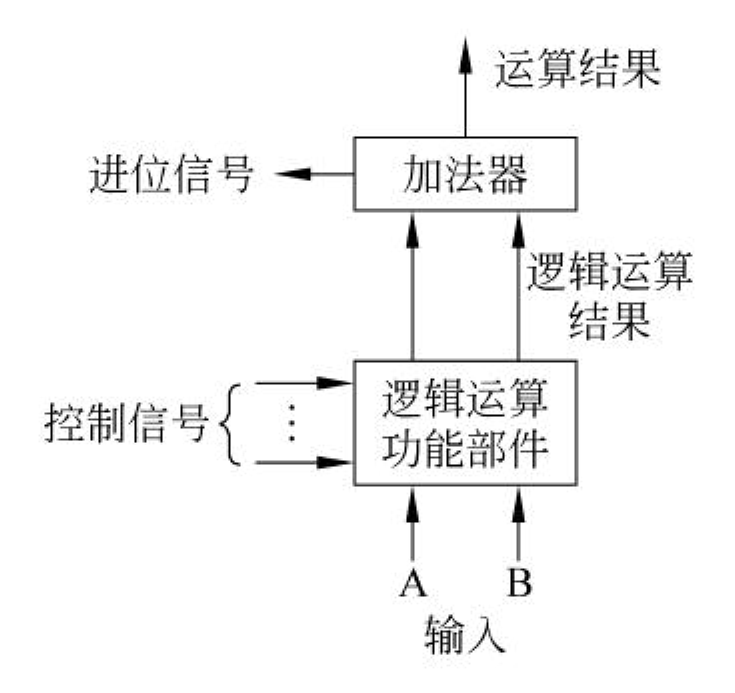
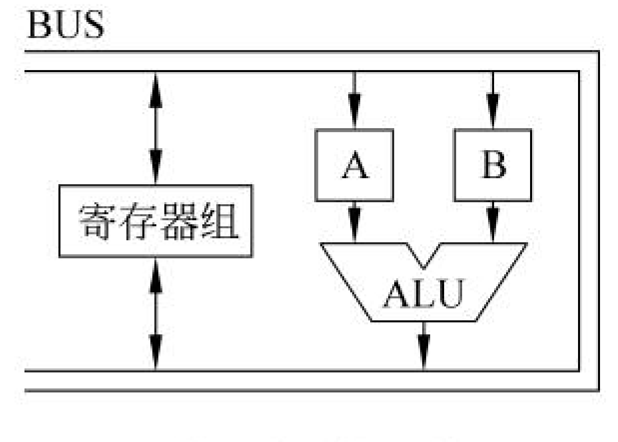
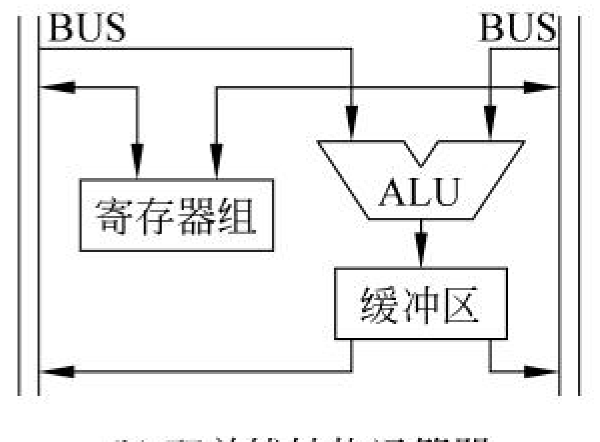


图 2-1 一位算术逻辑运算单元结构示意图

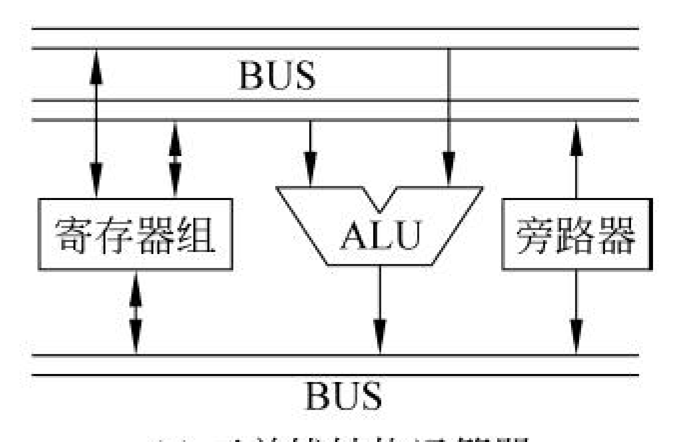
除了作为核心部件的 ALU 外,运算器中还有提供操作数和暂存中间运算结果及结果特征的寄存器及数据传送通道。CPU 内部用于传送数据和指令的传送通道称为 CPU 内部总线。运算器的结构根据其内部总线数量的不同分为 3 种,其示意图如图 2-2 所示。



(a) 单总线结构运算器



(b) 双总线结构运算器



(c) 三总线结构运算器

图 2-2 运算器结构示意图

1）单总线结构运算器

图 2-2(a)是单总线结构运算器的示意图，此时所有的部件都通过一条内部总线传递信息，任何时刻都只有一组数据从源部件传送到目标部件。由图中可看出，ALU 的输入端有两个用来暂时存放参加运算的操作数的锁存器。当要进行一次双操作数的运算时，首先通过总线将第一个操作数放入锁存器 A 或 B 中，然后再通过总线传送另一个操作数至另一个锁存器，之后进入 ALU 进行运算，运算的结果再通过总线置入某个内部通用寄存器。这种结构的控制简单，但速度比较慢。

2）双总线结构运算器

双总线结构是在运算器内部用两条总线来传送操作数的，如图 2-2(b)所示。此时参加运算的两个操作数可同时通过两条总线送至 ALU 进行运算，运算的结果经缓冲器再通过任意一条总线传送到通用寄存器。这种结构的运算器的处理速度显然就要比单总线结构的快。

3）三总线结构运算器

速度最快的运算器结构是图 2-2(c)所示的三总线结构。它用两条总线来传送操作数，一条专门用于传送运算结果。这样，在传送运算结果的同时就可通过另外两条总线传送参加操作数运算的操作数，只要 ALU 速度足够快，全部操作就可一步完成。

2.1.2 控制器

控制器的作用是控制程序的执行，它是整个系统的指挥中心，必须具备以下几项基本功能。

1）指令控制

计算机的工作过程就是连续执行指令的过程，指令在存储器中是连续存放的。一般情况下，按照顺序一条条地取出并执行指令，只有在碰到转移类指令时才会改变顺序。控制器要能根据指令所在的地址按顺序或在遇到转移指令时按照转移地址取出指令，分析指令(指令译码)，传送必要的操作数，并在指令执行结束后存放运算结果。总之，要保证计算机中的指令流的正常工作。

2）时序控制

指令的执行是在时钟信号的严格控制下进行的，一条指令的执行时间称为指令周期，不同指令的指令周期中所包含的机器周期数是不相同的，而一个机器周期中包含多少节拍(时钟周期)也不一定一样。这些时序信号用于计算机的工作基准，它们由控制器产生，使系统按一定的时序关系进行工作。

3）操作控制

操作控制是根据指令流程，确定在指令周期的各个节拍中要产生的微操作控制信号，以有效地完成各条指令的操作过程。

除此之外，控制器还要具有对异常情况及某些外部请求的处理能力，如出现运算溢出、中断请求等。

控制器的内部主要由以下几个部分组成。

(1) 程序计数器(Programming Counter，PC)。程序计数器用来存放下一条要执行

指令在存储器中的地址。在程序执行之前,应将程序的首地址(程序中第一条指令的地址)置入程序计数器。

(2) 指令寄存器(Instruction Register,IR)。指令寄存器用于存放从存储器中取出的待执行的指令。

(3) 指令译码器(Instruction Decoder,ID)。指令寄存器中待执行的指令须经过"翻译"才能明白要进行什么样的操作,即指令译码,这是指令译码器的主要功能。

(4) 时序控制部件。时序控制部件产生计算机工作中所需的各种时序信号。

(5) 微操作控制部件。这部分是控制器的主体。在计算机中,一条指令的功能是通过按一定顺序执行一系列基本操作来完成的。这些基本操作称为微操作,同时执行的一组微操作叫作微指令。例如 1 条加法指令就是由 4 条微指令解释执行的:取指微指令(包括的微操作有指令送地址总线、从存储器取指令送数据总线、指令送指令寄存器、程序计数器加 1)、计算地址微指令、取操作数微指令及加法运算并送结果微指令。

微操作控制部件用于产生与各条指令相对应的微操作。它根据当前正在执行的指令,在指令的各机器周期的各个节拍内产生相应的微操作控制信号,从而控制整个系统各部件的工作。

微处理器中控制器的一般结构示意图如图 2-3 所示。从图中可以看出,其中的核心部件是微操作控制部件。

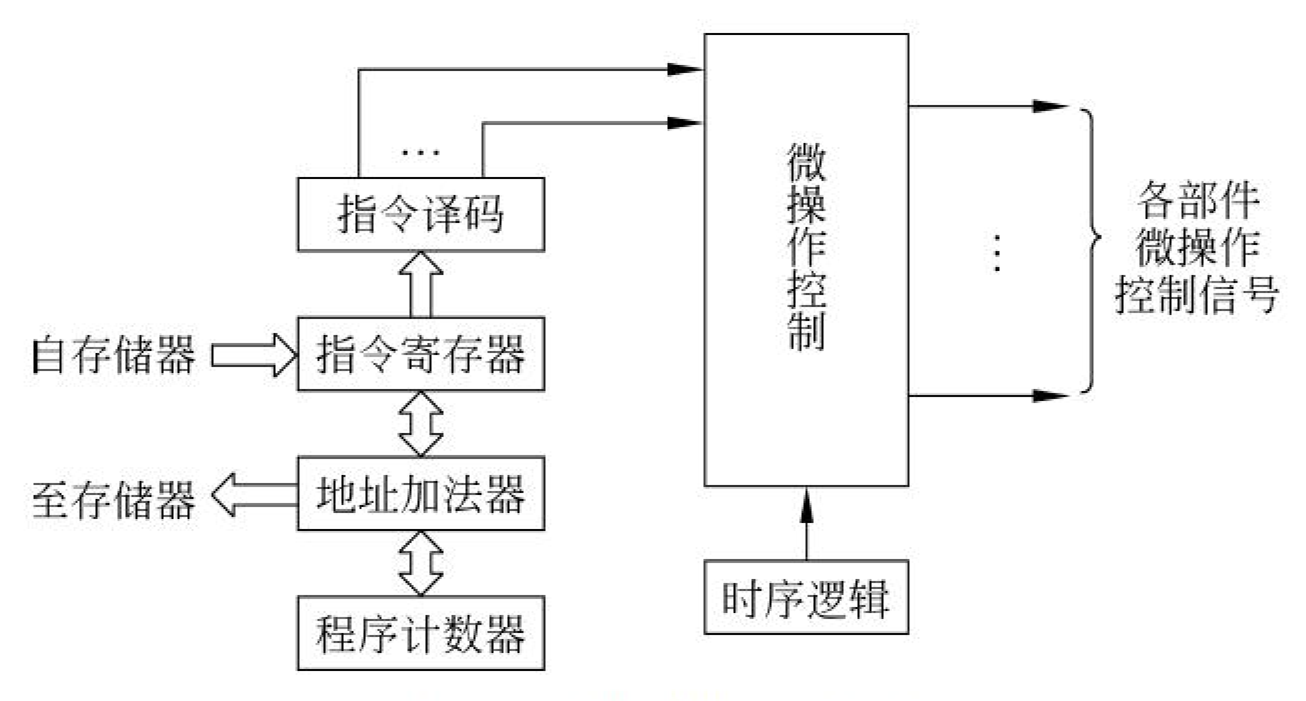


图 2-3　控制器结构示意图

2.2　8088/8086 微处理器

8088 是与 8086 同时代的微处理器,都属于第三代 CPU,它们具有完全相同的指令系统。在硬件结构上,8088 与存储器和 I/O 接口进行数据传输的外部数据总线宽度为 8 位,而 8086 的数据总线宽度为 16 位。除此之外,二者几乎没有什么差别,为其中一个 CPU 写的程序不需要任何修改就能在另一个 CPU 上运行。8086/8088 都具有 40 根外部引线,可以在单一 5V 电压下运行。由于这两种 CPU 的差异很小,所以本节以 8088 为主进行介绍。在没有特别指出时,所介绍的内容对两者均适用。

8088/8086 CPU 作为 IBM PC/XT 微型计算机的核心器件,为微机的发展做出了极其重要的贡献。

2.2.1 8088/8086 CPU 的特点

1. 8088/8086 的指令流水线

1.1.2 节已经学习，在程序的执行过程中，CPU 总是有规律地重复执行以下步骤：

① 从存储器中取出下一条指令；

② 指令译码(或分析指令)；

③ 如果指令需要，从存储器中读取操作数；

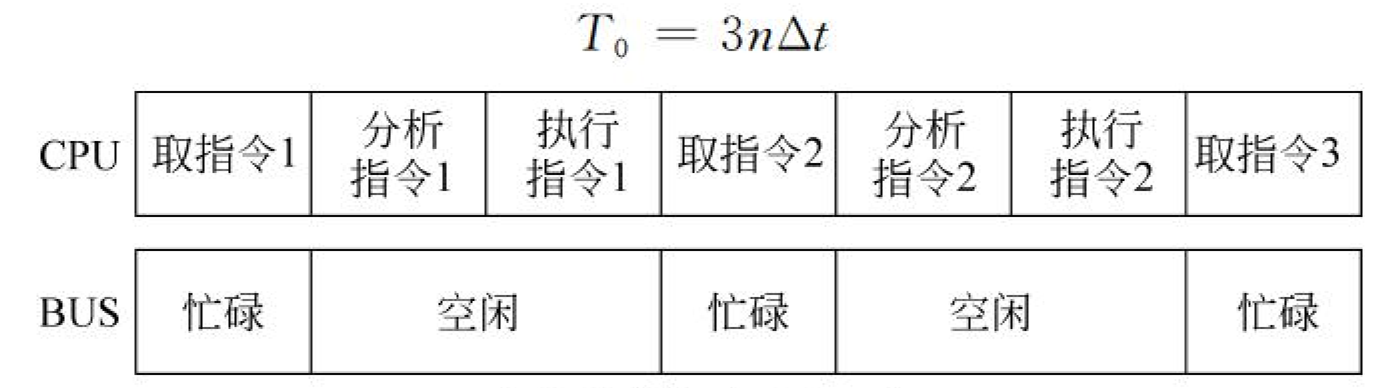
④ 执行指令(包括算术逻辑运算、I/O 操作、数据传送、控制转移等)；

⑤ 如果需要，将结果写入存储器。

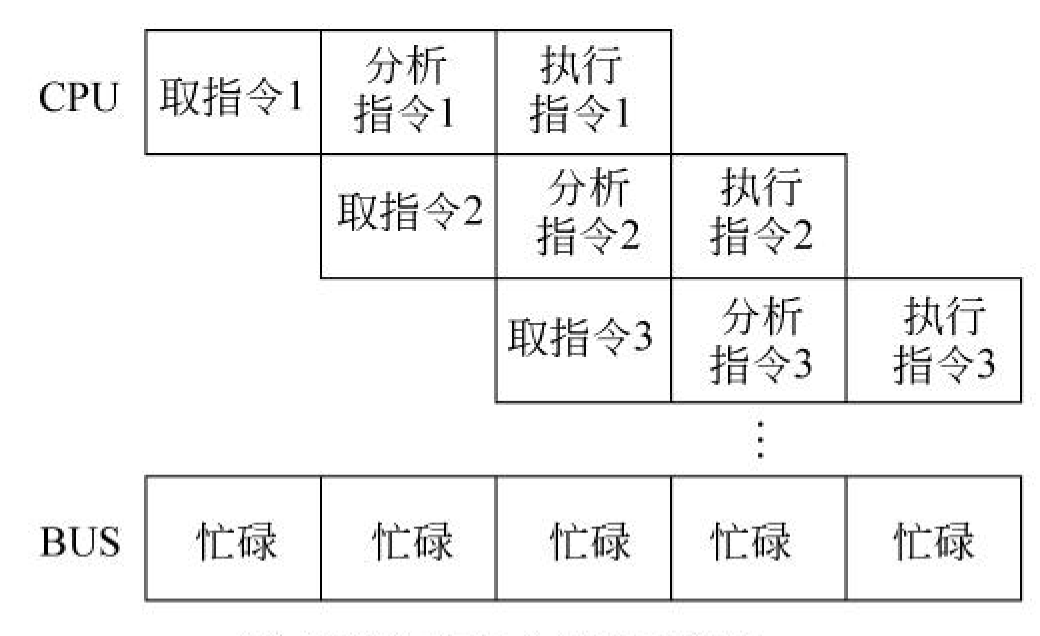
在 8088/8086 未出现以前，微处理器是按顺序串行完成以上各操作的。而从 8086/8088 开始，CPU 采用了一种新的结构来并行地完成这些工作。8088/8086 将上述步骤分配给 CPU 内两个独立的部件：执行单元(Execution Unit，EU)和总线接口单元(Bus Interface Unit，BIU)。EU 负责分析指令(指令译码)和执行指令，BIU 负责取指令、取操作数和写结果。这两个单元都能够独立地完成各自相应的工作。所以，当这两个单元并行工作时，在大多数情况下，取指令操作与执行指令操作都可重叠地进行。因为 BIU 已经从存储器中将 EU 要执行的指令“预取”了出来，所以大多数情况下“省掉”了取指令的时间，从而加快了程序的运行速度。

假设不考虑取操作数和写结果(有部分指令不需要这两个步骤)，将指令的执行过程简化为 3 个步骤，并假设这 3 个步骤所需时间完全相等(实际并不可能)，都为 Δt，则由图 2-4 知，采用顺序执行方式执行 n 条指令所需的时间为

$$T_0 = 3n\Delta t$$



(a) 顺序执行指令过程示意图



(b) 并行执行指令过程示意图

图 2-4 顺序执行和并行执行指令操作(流水线)

并行执行方式执行 n 条指令所需要的时间为

$$T = 3\Delta t + (n-1)\Delta t = (2+n)\Delta t$$

由此可见,采用并行执行方式所花费的时间及对总线的利用率都较顺序执行方式有较大的提高。这是 8088/8086 CPU 与其上代微处理器相比所具有的一大进步。这种并行操作的实现是因为在 8088/8086 CPU 内部(BIU 部分)设有一个指令预取队列,BIU 从内存中取出指令存放到指令预取队列,EU 再从指令队列中取出指令并执行。当 EU 从指令队列中取走指令,指令队列出现空字节时,BIU 就自动执行一次取指令周期,从内存中取出后续的指令代码放入队列中;如果遇到跳转指令,BIU 会使指令队列复位,从新地址中重新取出指令,并立即传给 EU 去执行。

指令队列的存在使 8086/8088 的 EU 和 BIU 能够并行工作,从而减少了 CPU 为取指令而等待的时间,提高了 CPU 的执行效率和运行速度。另外也降低了对存储器存取速度的要求。

当然,这种并行流水线结构不能与现在新型 CPU(如 Pentium、K7 等)的指令流水线相提并论,但它为现代流水线技术奠定了基础,也使 8086/8088 成为 CPU 发展史上的一个里程碑。

2. 内存的分段管理技术

8088/8086 CPU 的内部结构都是 16 位的,即内部的寄存器只能存放 16 位二进制码,内部的总线同时也只能传送 16 位二进制码。16 位二进制码最多只具有 $2^{16}=64K$ 种组合。如果用二进制码表示地址(计算机中只能识别二进制),则 8088/8086 就只能产生 64K 个地址,亦即最多能够管理 64 个内存单元。

由于内存容量的大小对计算机的性能有直接的影响,为了提高系统的执行速度,人们希望尽可能地提高系统管理(寻址)内存的能力。为此,8086/8088 采用了分段管理的方法,将内存地址空间分为多个逻辑段,每个逻辑段最大为 64K 个单元,段内每个单元的地址码(称为偏移地址或相对地址)长度为 16 位,满足其 16 位内部结构的要求;再为每个段设置段地址(也称段基地址),以区分不同的逻辑段。

所以,8088/8086 系统中,内存每个单元的地址都由两部分组成,即段地址和段内偏移地址。这就相当于一栋大楼中的每一个房间的编号都是由楼层号和在所在层的位置号(相对于起始房间的位置)组成的。例如,312 房间通常表示 3 楼第 12 号房间。

8088/8086 CPU 内部具有专门存放段地址的段寄存器和存放偏移地址的地址寄存器,将两类不同寄存器的内容送入地址加法器中合成,就形成了指向内存某一具体单元的地址(物理地址)。有关物理地址的详细内容参见 2.2.4 节。

3. 支持多处理器系统

8086/8088 具有最小和最大两种工作模式以及内置的多任务处理能力,可通过模式选择引脚进行选择。

(1) 最小模式也称为单处理器模式。此时 CPU 仅支持由少量设备组成的单处理器系统而不支持多处理器结构,系统控制总线的信号由 8088 CPU 直接产生,且构成的系统

不能进行 DMA 传送。

(2) 最大模式也称为多处理器模式。此时 CPU 能支持系统总线上的多个处理器，由总线控制器提供所有总线控制信号和命令信号。

2.5.4 节将进一步介绍有关 8088 CPU 工作于最大模式和最小模式时的系统结构。

2.2.2 8088 CPU 的外部引脚及其功能

8088 和 8086 CPU 都是具有 40 条引出线的集成电路芯片，采用双列直插式封装，图 2-5 是 8088 处理芯片的引脚图，8086 与之基本相同。为了减少芯片的引线，8088 的许多引脚具有双重功能，采用分时复用方式工作，即在不同时刻，这些引线上的信号是不相同的。同时，8088 的最大和最小两种工作模式可以通过在 $MN/\overline{MX}$输入引脚加上不同的电平来进行选择。当 $MN/\overline{MX}=1$ 时，8088 工作在最小模式，此时，构成的微型机中只包括一个 8088 处理器，且系统总线由 8088 的引线直接引出形成；当 $MN/\overline{MX}=0$ 时，8088 工作在最大模式，在此模式下，构成的微型计算机中除了有 8088 CPU 之外，还可以接另外的处理器(如 8087 数字协处理器)构成多微处理器系统。在最大模式下，微机的系统总线要由 8088 和总线控制器(8288)共同形成。图 2-5 中括号内的引脚信号用于最大模式。

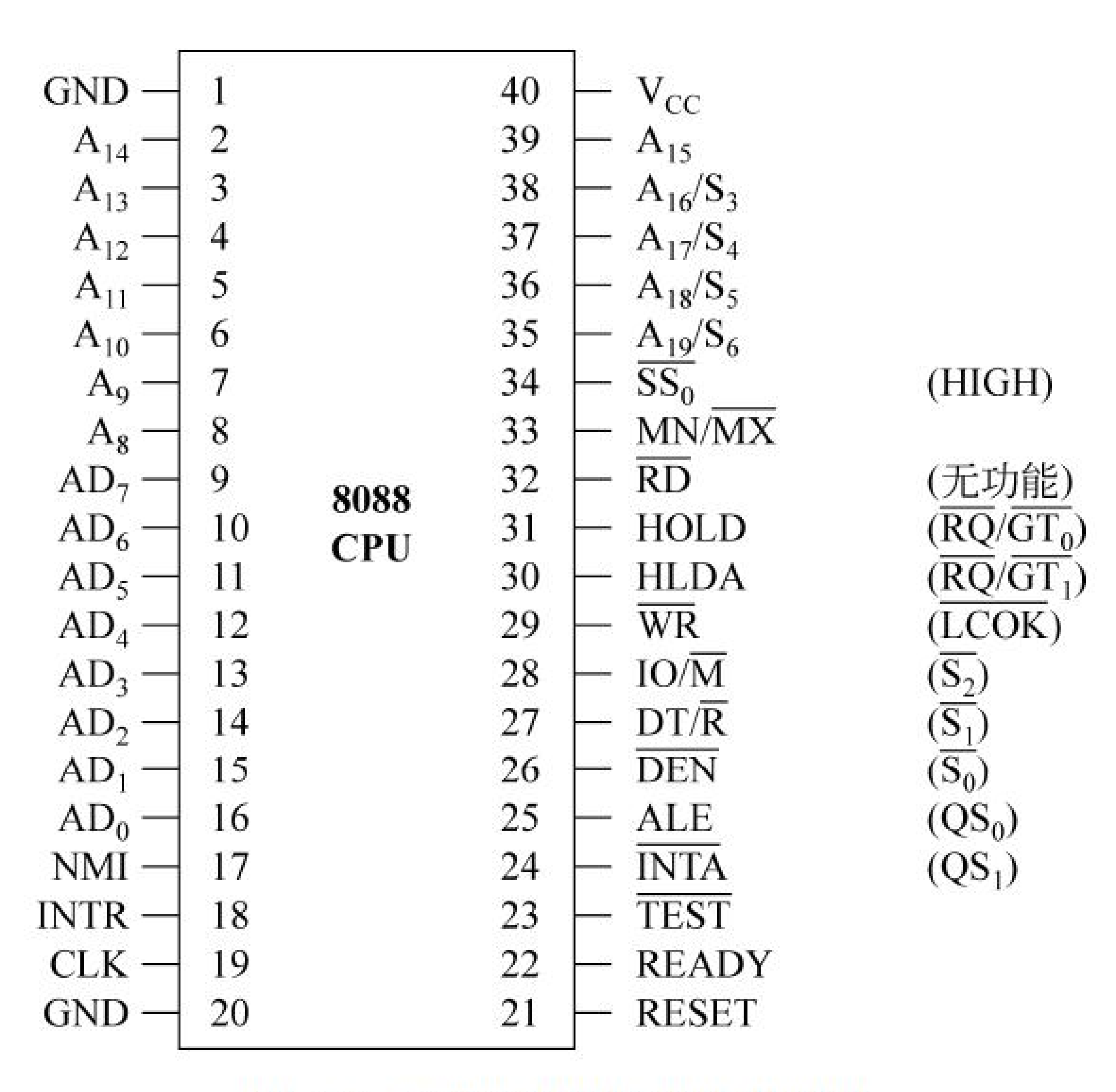


图 2-5 8088 处理器芯片引脚图

1. 最小模式下的引脚

在最小模式下，8088 的引脚定义如下。

(1) $A_{16}\sim A_{19}/S_3\sim S_6$：地址、状态复用的引脚，三态输出。在 8088 执行指令过程中，某一时刻从这 4 个引脚上送出地址的最高 4 位 $A_{16}\sim A_{19}$；而在另外时刻，这 4 个引脚送出状态信号 $S_3\sim S_6$。这些状态信号里，S_6 恒等于 0，S_5 指示中断允许标志位 IF 的状态，S_4、

S_3 的组合指示 CPU 当前正在使用的段寄存器,其编码如表 2-1 所示。

表 2-1 段寄存器状态线

S_4	S_3	当前正在使用的段寄存器	S_4	S_3	当前正在使用的段寄存器
0	0	ES	1	0	CS 或未使用任何段寄存器
0	1	SS	1	1	DS

(2) A_8～A_{15}:中 8 位地址信号,三态输出。CPU 寻址内存或接口时,从这些引脚送出地址 A_8～A_{15}。

(3) AD_0～AD_7:地址、数据分时复用的双向信号线,三态。当 ALE=1 时,这些引脚上传输的是地址信号;当$\overline{DEN}$=0 时,这些引脚上传输的是数据信号。

(4) IO/$\overline{M}$:输入输出/存储器控制信号,三态。IO/$\overline{M}$ 引脚用来区分当前操作是访问存储器还是访问 I/O 端口。若此引脚输出为低电平,访问存储器;若输出为高电平,则是访问 I/O 端口。

(5) $\overline{WR}$:写信号输出,三态。此引脚输出为低电平时,表示 CPU 正在对存储器或 I/O 端口进行写操作。

(6) DT/$\overline{R}$:数据传送方向控制信号,三态。DT/$\overline{R}$ 引脚用于确定数据传送的方向。高电平时,CPU 向存储器或 I/O 端口发送数据;低电平时,CPU 从存储器或 I/O 接口接收数据。此信号用于控制总线收发器 8286/8287 的传送方向。

(7) $\overline{DEN}$:数据允许信号,三态。该信号有效时,表示数据总线上具有有效数据。它在每次访问内存或 I/O 接口以及在中断响应期间有效,常用作数据总线驱动器的片选信号。

(8) ALE:地址锁存信号,三态输出,高电平有效。当它为高电平时,表明 CPU 地址线上有有效地址。因此,它常作为锁存控制信号将 A_0～A_{19} 锁存到地址锁存器。

(9) $\overline{RD}$:读选通信号,三态输出,低电平有效。当其有效时,表示 CPU 正在对存储器或 I/O 接口进行读操作。

(10) READY:外部同步控制输入信号,高电平有效。它是由被访问的内存或 I/O 设备所发出的响应信号,当其有效时,表示存储器或 I/O 设备已准备好,CPU 可以进行数据传送。

若存储器或 I/O 设备没有准备好,则使 READY 信号为低电平。CPU 在 T_3 周期采样 READY 信号,若其为低,CPU 自动插入等待周期 T_w(1 个或多个),直到 READY 变为高电平后 CPU 才脱离等待状态,完成数据传送过程。

(11) INTR:可屏蔽中断请求输入信号,高电平有效。CPU 在每条指令的最后一个周期采样该信号,以决定是否进入中断响应周期。这个引脚上的中断请求信号可用软件屏蔽。

(12) $\overline{TEST}$:测试信号输入引脚,低电平有效。当 CPU 执行 WAIT 指令时,每隔 5 个时钟周期对此引脚进行一次测试,若为高电平,CPU 则处于空转状态进行等待;当该引脚变为低电平时,CPU 结束等待状态,继续执行下一条指令。

(13) NMI:非屏蔽中断请求输入信号,上升沿触发。这个引脚上的中断请求信号不能用软件屏蔽,CPU 在当前指令执行结束后就进入中断过程。

(14) RESET：系统复位输入信号，高电平有效。为使 CPU 完成内部复位过程，该信号至少要在 4 个时钟周期内保持有效。复位后 CPU 内部寄存器的状态如表 2-2 所示。当 RESET 返回低电平时，CPU 将重新启动。

表 2-2　复位后 CPU 的内部寄存器状态

内部寄存器	内　容	内部寄存器	内　容
CS	FFFFH	IP	0000H
DS	0000H	FLAGS	0000H
SS	0000H	其余寄存器	0000H
ES	0000H	指令队列	空

(15) $\overline{\text{INTA}}$：中断响应信号输出，低电平有效。此信号是 CPU 对中断请求信号 INTR 的响应。在响应过程中，CPU 在$\overline{\text{INTA}}$引脚上连续输出两个负脉冲用作外部中断源的中断向量码的读选通信号。

(16) HOLD：总线保持请求信号输入，高电平有效。当某一总线主控设备要占用系统总线时，通过此引脚向 CPU 提出请求。

(17) HLDA：总线保持响应信号输出，高电平有效。这是 CPU 对 HOLD 请求的响应信号，当 CPU 收到有效的 HOLD 信号后，就会对其做出响应：一方面使 CPU 的所有三态输出的地址信号、数据信号和相应的控制信号变为高阻状态(浮动状态)；同时输出一个有效的 HLDA，表示处理器现在已放弃对总线的控制。当 CPU 检测到 HOLD 信号变低后，就立即使 HLDA 变低，同时恢复对总线的控制。

(18) $\overline{\text{SS}_0}$：系统状态信号输出。它与 IO/$\overline{\text{M}}$ 和 DT/$\overline{\text{R}}$ 信号决定了最小模式下当前总线周期的状态。三者组合所表示的处理器操作见附录 B(B.1)。

(19) CLK：时钟信号输入引脚。8088 的标准时钟频率为 4.77MHz。

(20) V_{CC}：5V 电源输入引脚。

(21) GND：地线。

2. 最大模式下的引脚

当 MN/$\overline{\text{MX}}$加上低电平时，8088 CPU 工作在最大模式下。此时，除引脚 24 到 34 外，其他引脚与最小模式完全相同，如图 2-7 中括号内的引脚信号。

(1) $\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$：总线周期状态信号，低电平有效，三态输出。它们连接到总线控制器 8288 的输入端，8288 对它们译码后可以产生系统总线所需要的各种控制信号。$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$ 的代码组合以及对应的操作见附录 B(B.2)。

(2) $\overline{\text{RQ}}/\overline{\text{GT}_1}$、$\overline{\text{RQ}}/\overline{\text{GT}_0}$：总线请求/总线响应信号引脚。每一个引脚都具有双向功能，既是总线请求输入也是总线响应输出。但是$\overline{\text{RQ}}/\overline{\text{GT}_0}$比$\overline{\text{RQ}}/\overline{\text{GT}_1}$优先级高。这些引脚内部都有上拉电阻，所以在不使用时可以悬空。两个引脚的功能如下。

当其他的总线控制设备要使用系统总线时，会产生一个总线请求信号(一个时钟周期宽的负脉冲)，并把它送到$\overline{\text{RQ}}/\overline{\text{GT}}$引脚，类似于最小模式下的 HOLD 信号。CPU 检测到总线请求信号后，在下一个 T_4 或 T_1 期间，在$\overline{\text{RQ}}/\overline{\text{GT}}$引脚送出总线响应信号(一个时钟

周期宽的负脉冲)给请求总线的设备,它类似于最小模式下的 HLDA 信号。然后从下一个时钟周期开始,CPU 释放总线。总线请求设备使用完总线后,再产生一个$\overline{RQ}/\overline{GT}$信号。CPU 检测到该信号后,从下一个时钟周期开始重新控制总线。

(3) $\overline{LOCK}$:总线封锁信号输出,低电平有效。该信号有效时,CPU 锁定总线,不允许其他总线控制设备申请使用系统总线。$\overline{LOCK}$信号由前缀指令 LOCK 产生,LOCK 指令后面的一条指令执行完后,该信号失效。

(4) QS_1、QS_0:指令队列状态输出。根据该状态信号,从外部可以跟踪 CPU 内部的指令队列。QS_1、QS_0 的编码见附录 B(B.3)。

(5) HIGH:在最大模式下始终为高电平输出。

此外,在最大模式下,$\overline{RD}$引脚不再使用。

2.2.3 8088/8086 CPU 的功能结构

1. 8088/8086 CPU 的内部结构

8086 与 8088 结构极为相似,都是由执行单元 EU 和总线接口单元 BIU 两大部分构成。图 2-6 给出了 8088 微处理器的内部结构框图。

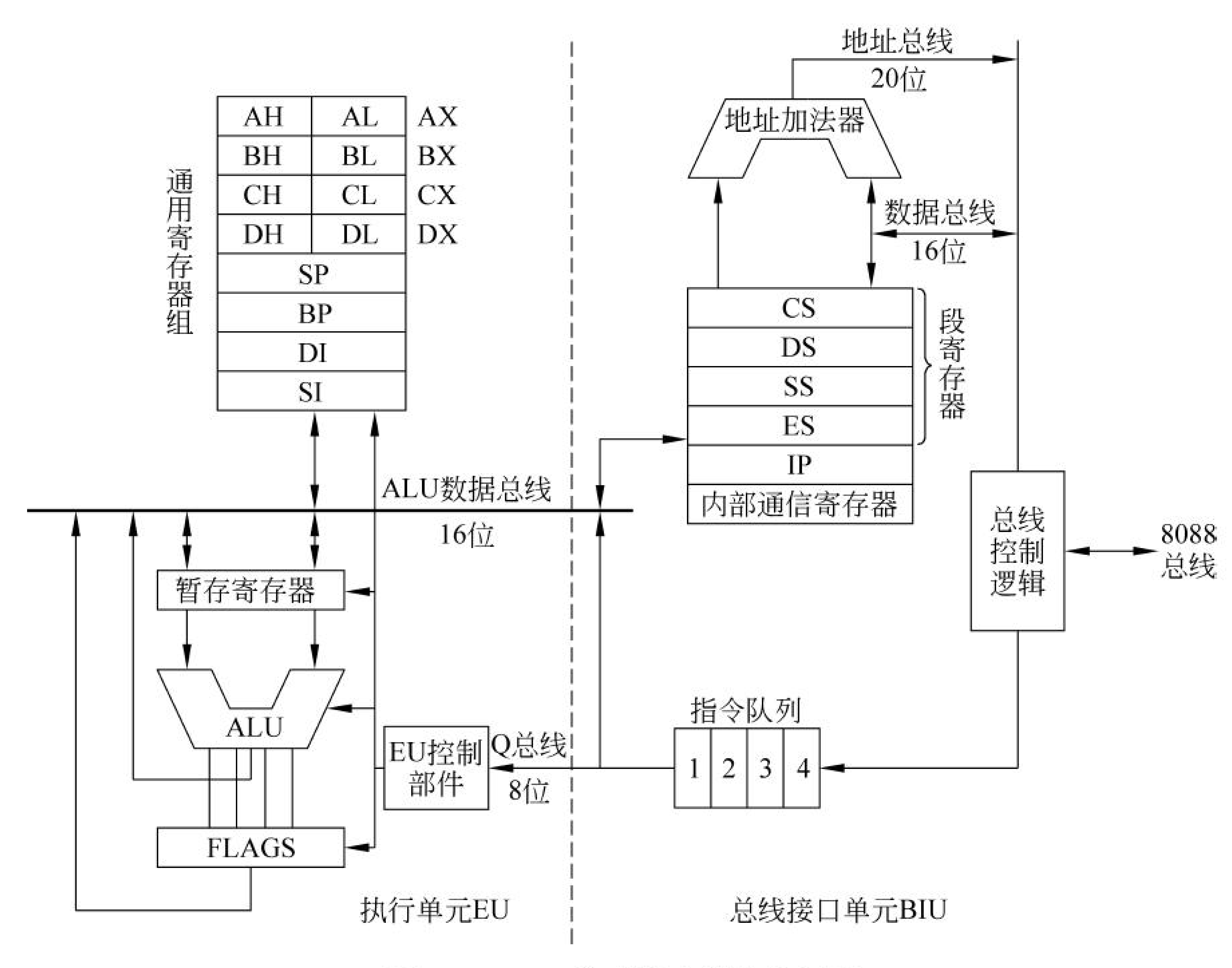


图 2-6　8088 处理器内部结构框图

执行单元 EU 的主要功能是:执行指令、分析指令、暂存中间运算结果并保留结果的特征。它由算术逻辑单元(运算器)ALU、通用寄存器、标志寄存器和 EU 控制电路组成。

EU 在工作时不断地从指令队列中取出指令代码，对其译码后产生完成指令所需要的控制信息。数据在 ALU 中进行运算，运算结果的特征保留在标志寄存器 FLAGS 中。

总线接口单元 BIU 负责 CPU 与存储器、I/O 接口之间的信息传送。它由段寄存器、指令指针寄存器、指令队列、地址加法器以及总线控制逻辑组成。8088 的指令队列长度为 4 字节，8086 的指令队列长度为 6 字节。

当 EU 从指令队列中取走指令，指令队列出现空字节时，BIU 就自动执行一次取指令周期，从内存中取出后续的指令代码放入队列中。当 EU 需要数据时，BIU 根据 EU 给出的地址从指定的内存单元或外设中取出数据供 EU 使用。在运算结束时，BIU 将运算结果送入指定的内存单元或外设。如果指令队列为空，EU 就等待，直到有指令为止。若 BIU 正在取指令，EU 发出访问总线的请求，则必须等 BIU 取指令完毕后该请求才能得到响应。一般情况下，程序顺序执行，当遇到跳转指令时，BIU 就使指令队列复位，从新地址取出指令，并立即传给 EU 去执行。

指令队列的存在使 8086/8088 的 EU 和 BIU 并行工作，从而减少了 CPU 为取指令而等待的时间，提高了 CPU 的利用率，加快了整机的运行速度，另外也降低了对存储器存取速度的要求。

BIU 中的地址加法器用来产生 20 位的物理地址。8086/8088 的寄存器都是 16 位的，无法装载 20 位的物理地址。为了解决这个问题，8086/8088 采用了将地址空间分段的方法，即将 2^{20}(1MB)的地址空间分为若干个 64KB 的段，然后用段基址加上段内偏移来访问物理存储器。8086/8088 规定，分段总是从 16 字节的边界处开始，所以段的起始地址最低 4 位总是 0，即××××0H，这样每个段的基地址只需用 16 位便可表示。也就是说，段基址实际上是段起始地址的高 16 位。由于段基址的这个特点，BIU 在计算存储器的物理地址时，即是将段基址左移 4 位然后与段内偏移相加，如图 2-7 所示。

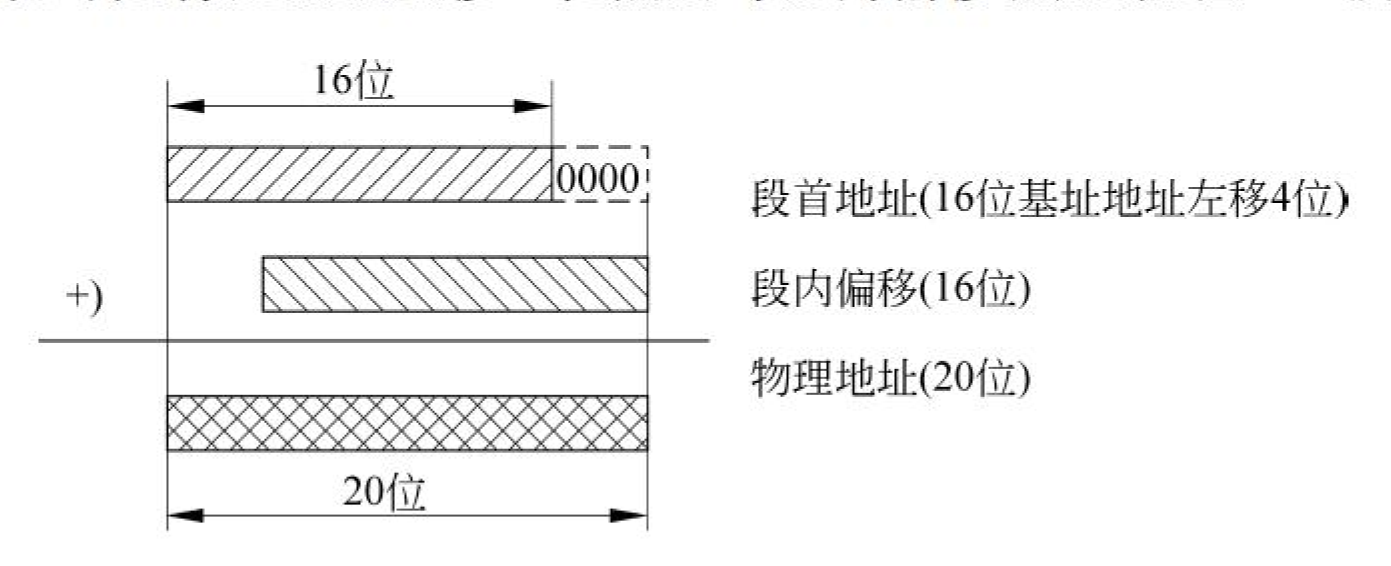


图 2-7　物理地址的生成方法

2. 8088/8086 CPU 的内部寄存器

8086/8088 CPU 内部共有 14 个 16 位寄存器。按其功能可分为三大类，即通用寄存器(8 个)、段寄存器(4 个)、控制寄存器(2 个)，如图 2-8 所示。

1) 通用寄存器

通用寄存器包括数据寄存器、地址指针寄存器和变址寄存器。

(1) 数据寄存器 AX、BX、CX、DX。数据寄存器一般用于存放参与运算的数据或运算的结果。每一个数据寄存器都是 16 位寄存器，但又可将高、低 8 位分别作为两个独立

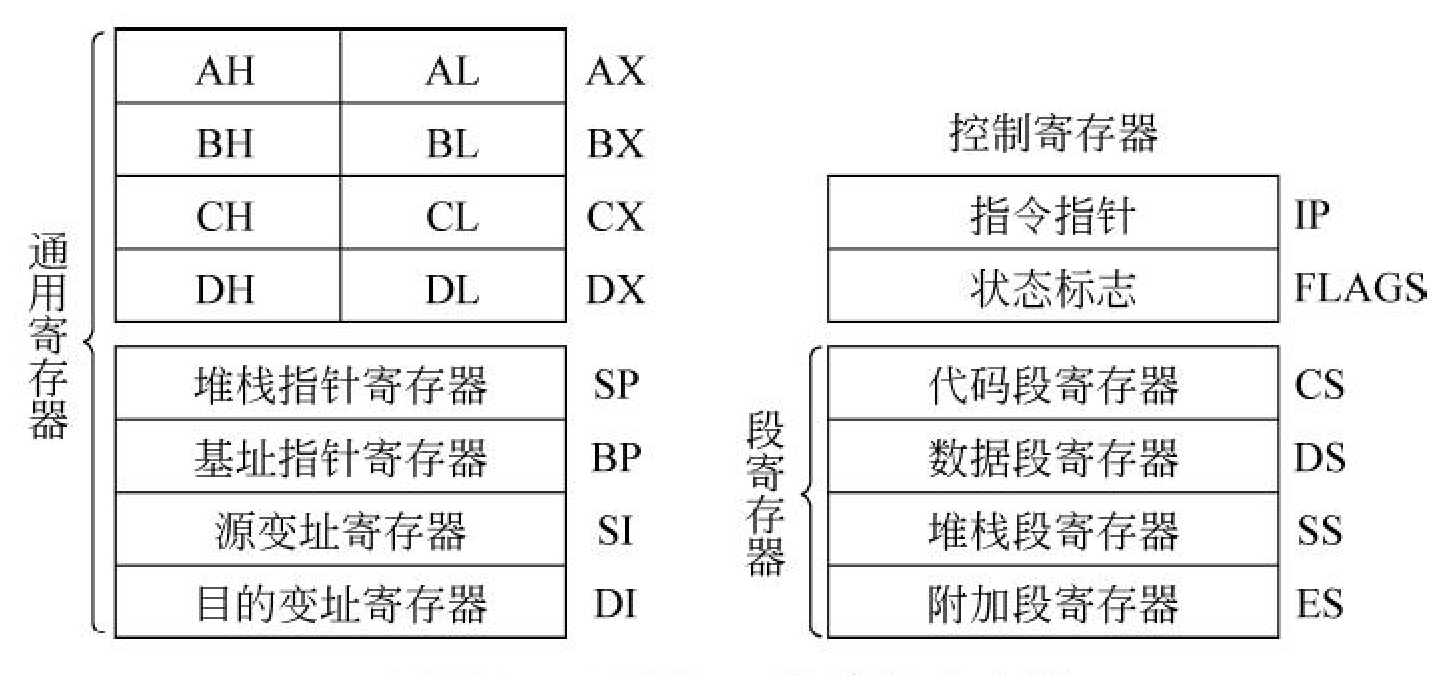


图 2-8　8088/8086 的内部寄存器

的 8 位寄存器使用。它们的高 8 位记作 AH、BH、CH、DH，低 8 位记作 AL、BL、CL、DL。这种灵活的使用方法给编程带来了极大的方便，既可以处理 16 位数据，也能处理 8 位数据。

数据寄存器除了作为通用寄存器使用外，它们还有各自的习惯用法。

① AX(Accumulator)：累加器，常用于存放算术逻辑运算中的操作数，另外所有的 I/O 指令都使用累加器与外设接口传送信息。

② BX(Base)：基址寄存器，常用来存放访问内存时的基地址。

③ CX(Count)：计数寄存器，在循环和串操作指令中用作计数器。

④ DX(Data)：数据寄存器，在寄存器间接寻址的 I/O 指令中存放 I/O 端口的地址。

另外，在做双字长乘除法运算时，DX 与 AX 合起来存放一个双字长数(32 位)，其中 DX 存放高 16 位，AX 存放低 16 位。

(2) 地址指针寄存器 SP、BP。

① SP(Stack Pointer)：堆栈指针寄存器，它在堆栈操作中用来存放栈顶偏移地址，永远指向堆栈的栈顶。

② BP(Base Pointer)：基址指针寄存器。一般也常用来存放访问内存时的基地址，但它通常与 SS 寄存器配对使用。(比较：BX 通常与 DS 寄存器配对使用。)

作为通用寄存器，SP 和 BP 也可以存放数据。但实际上，它们更经常更重要的用途是存放内存单元的偏移地址，特别是 SP 在访问堆栈时作为指向堆栈栈顶的指针。

(3) 变址寄存器 SI、DI。SI(Source Index)称为源变址寄存器，DI(Destination Index)称为目的变址寄存器，它们常常在变址寻址方式中作为索引指针。

2) 段寄存器 CS、SS、DS、ES

CS(Code Segment)称为代码段寄存器，SS(Stack Segment)称为堆栈段寄存器，DS(Data Segment)称为数据段寄存器，ES(Extra Segment)称为附加数据段寄存器。段寄存器用于存放段基址，即段起始地址的高 16 位。

3) 控制寄存器 IP、FLAGS

IP(Instruction Pointer)称为指令指针寄存器，用以存放预取指令的偏移地址。CPU 取指令时总是以 CS 为段基址，以 IP 为段内偏移地址。当 CPU 从 CS 段中偏移地址为(IP)的内存单元中取出指令代码的一个字节后，IP 自动加 1，指向指令代码的下一个字节。用户程序不能直接访问 IP。

FLAGS 称为标志寄存器或程序状态字(PSW),它是 16 位寄存器,但只使用其中的 9 位。这 9 位包括 6 个状态标志和 3 个控制标志,如图 2-9 所示。

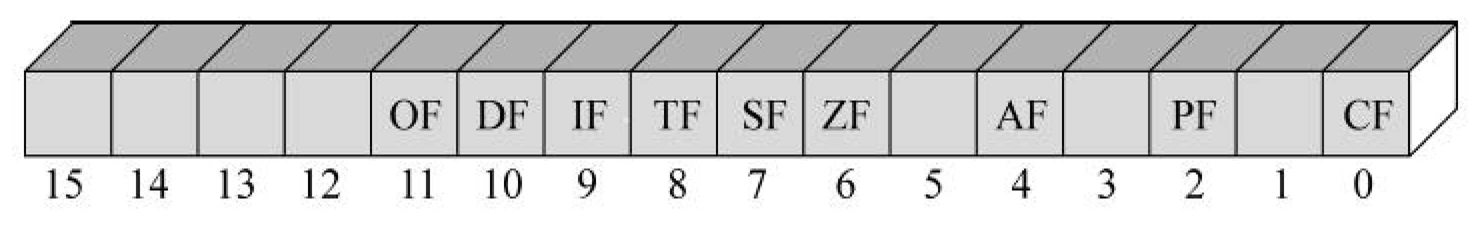


图 2-9 8088/8086 的标志寄存器

(1) 状态标志位记录了算术和逻辑运算结果的一些特征,如:结果是否为 0,是否有进位、借位,结果是否溢出等。不同指令对标志位具有不同的影响。

① CF:进位标志位。当进行加(减)法运算时,若最高位向前有进(借)位,则 CF=1,否则 CF=0。

② PF:奇偶标志位。当运算结果的低 8 位中 1 的个数为偶数时 PF=1,为奇数时 PF=0。

③ AF:辅助进位标志位。在加(减)法操作中,D_3 向 D_4 有进位(借位)发生时,AF=1,否则 AF=0。DAA 和 DAS 指令测试这个标志位,以便在 BCD 加法或减法之后调整 AL 中的值。

④ ZF:零标志位。当运算结果为零时 ZF=1,否则 ZF=0。

⑤ SF:符号标志位。当运算结果的最高位为 1 时 SF=1,否则 SF=0。

⑥ OF:溢出标志位。当算术运算的结果超出了带符号数的范围,即溢出时 OF=1,否则 OF=0。

(2) 控制标志位用于设置控制条件。控制标志被设置后便对其后的操作产生控制作用。

① TF:陷阱标志位。当 TF=1 时,激活处理器的调试特性,使 CPU 处于单步执行指令的工作方式。每执行一条指令后,自动产生一次单步中断,从而使用户能逐条指令地检查程序。

② IF:中断允许标志位。IF=1 使 CPU 可以响应可屏蔽中断请求。IF=0 使 CPU 禁止响应可屏蔽中断请求。IF 的状态对不可屏蔽中断及内部中断没有影响。

③ DF:方向标志位。方向标志位在执行串操作指令时控制操作的方向。DF=1 时按减地址方式进行,即从高地址开始,每进行一次操作,地址指针自动减 1(或减 2);DF=0 时则按增地址方式进行。

2.2.4 8088/8086 CPU 的存储器组织

1. 物理地址与逻辑地址

8088/8086 有 20 条地址线,可寻址的最大物理内存容量为 1MB(2^{20}),其中任何一个内存单元都有一个 20 位的地址,称为内存单元的物理地址。前面已经介绍过,8088/8086 内部寄存器都只有 16 位,而访问内存单元在多数情况下都要通过寄存器间接寻址,很明显,若不采取特殊措施,是无法访问 1MB 的存储空间的。8086/8088 采用了将地址空间

分段的方法来解决这个问题,即将 1MB 的地址空间分为若干个 64KB 的段,然后用段基地址加上段内偏移地址来访问物理存储器。

段基地址和段内偏移地址又称为逻辑地址,逻辑地址通常写成 XXXXH:YYYYH 的形式,其中 XXXXH 是段基址,YYYYH 是段内偏移地址(也称为相对地址)。如图 2-7 所示,20 位的物理地址与逻辑地址的关系如下:

$$物理地址=段基址\times16+段内偏移$$

段基地址乘以 16 相当于段基地址左移 4 位(或段基地址后面加 4 个 0),然后再与偏移地址相加,即可得到 20 位的物理地址。例如,逻辑地址 3A00H:12FBH 对应的物理地址是 3B2FBH。

因为 8086/8088 CPU 中有 4 个段寄存器,所以它同时可以访问 4 个存储段。段与段之间可以重合、重叠、紧密连接或间隔分开。

分段(段加偏移)寻址所带来的好处是允许程序在存储器内重定位(浮动),允许实模式下编写的程序在保护模式下运行。可重定位程序是一个不加修改就可以在任何存储区域中运行的程序。这是因为段内偏移总是相对段起始地址(段首地址)的,所以只要在程序中不使用绝对地址访问存储器,就可以把整个程序作为一个整体移到一个新的区域。在 DOS 中,程序载入到内存时由操作系统来指定段寄存器的内容,以实现程序的重定位。

2. 段寄存器的使用

段寄存器的设立不仅使 8088 的存储空间扩大到 1MB,而且为信息按特征分段存储带来了方便。在存储器中,信息按特征可分为程序代码、数据、堆栈等。为了操作方便,存储器可以相应地划分为:程序段——用来存放程序的指令代码;数据段及附加数据段——用来存放数据和运算结果;堆栈段——用来传递参数、保存数据和状态信息。有时一种类型的段可能还会有多个。通过修改段寄存器的内容,就可将这些段设置在存储器的任何位置上。这些段可以通过段寄存器的设置使之相互独立,也可将它们部分或完全重叠。

8088/8086 对访问不同内存段所使用的段寄存器和相应的偏移地址的来源有一些具体约定,如表 2-3 所示。

表 2-3　8088/8086 对段寄存器使用的约定

序号	内存访问类型	默认段寄存器	可重设的段寄存器	段内偏移地址来源
1	取指令	CS	无	IP
2	堆栈操作	SS	无	SP
3	串操作之源串	DS	ES、SS	SI
4	串操作之目标串	ES	无	DI
5	BP 用作基址寻址	SS	ES、DS	按寻址方式计算得有效地址
6	一般数据存取	DS	ES、SS	按寻址方式计算得有效地址

根据表 2-3,访问存储器时,其段地址可以由“默认”的段寄存器提供,也可以由“指定”的段寄存器提供。当指令中没有显式地“指定”使用某一个段寄存器时,就由“默认”段寄存器来提供访问内存的段地址。在实际进行程序设计时,大多数情况都用默

认段寄存器来寻址内存。在 3、5、6 这 3 种访问存储器操作中，允许在指令中指定使用另外的段寄存器，这样可很灵活地访问不同的内存段。这种指定通常是靠在指令码中增加一个字节的前缀来实现。1、2、4 这 3 种类型的内存访问只能用默认的段寄存器，即取指令一定要使用 CS；堆栈操作一定要使用 SS；串操作指令的目的段基地址一定要用 ES。

DS、ES 和 SS 要用传送指令来进行设置，但在用户程序中不允许设置 CS，CS 一般由操作系统进行设置。宏汇编语言中的伪指令 ASSUME 及 JMP、CALL、RET、INT 和 IRET 等指令可以改变和影响 CS 的内容。更改段寄存器的内容意味着内存段的移动，这说明无论程序段、数据段、附加段还是堆栈段都可以用重设段寄存器内容的方法来改变逻辑段在内存中的位置。

有时也把一个存储器段用指向它的段寄存器的名字来表示。例如，如果一个数据段的段基址由 DS 来指明，这个段就可称为 DS 段；同理，若段基地址既在 DS 中，又在 ES 中，则该段既可以称为 DS 段，也可以称为 ES 段。

表中前四类内存操作的偏移地址只能使用一个 16 位的指针寄存器或变址寄存器。例如，取指令时为指令指针寄存器 IP；堆栈操作时为堆栈指针 SP；串操作时分别为 SI 和 DI。后两类内存操作则根据不同的寻址方式来计算出段内偏移。

2.2.5　8088/8086 CPU 的工作时序

工作时序表征微处理器各引脚在时间上的工作关系。时序可分为两种不同的粒度：时钟周期和总线周期。一条指令的执行需要若干个总线周期才能完成，而一个总线周期又由若干个时钟周期构成。

微处理器在运行过程中是按照一个统一的时钟一步步地执行每一个操作的，每个时钟脉冲的持续时间就称为一个时钟周期。显然，时钟周期越短，CPU 执行的速度就越快。

在 8088 CPU 中，CPU 与内存或接口间都通过总线进行通信，如将一个字节写入内存单元中或者从内存某单元中读一个字节到 CPU，这种通过总线进行一次读(或)写的过程称为一个总线周期，一个总线周期包括多个时钟周期。典型的总线周期如图 2-10 所示。

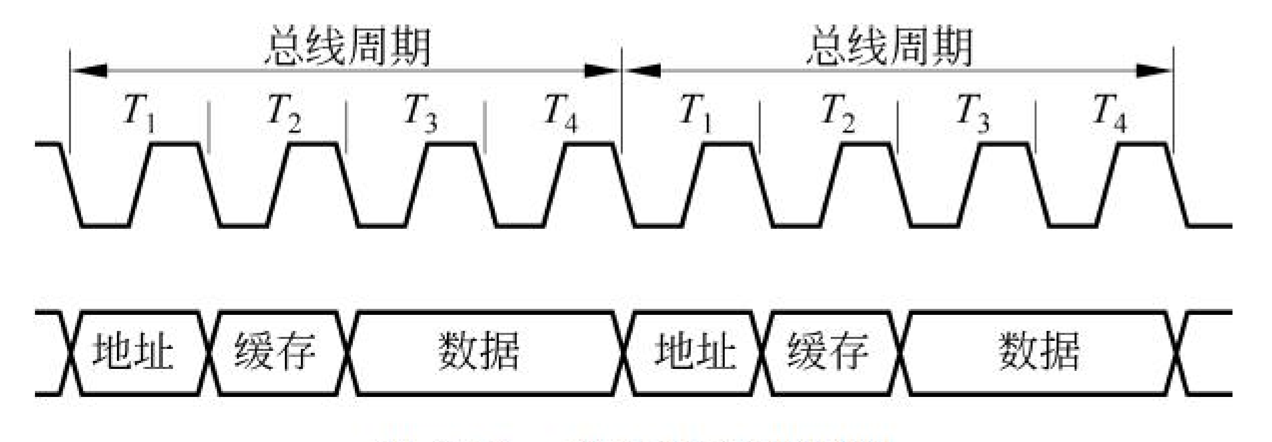


图 2-10　典型的总线周期

下面简要介绍一下 8088 CPU 在最小模式下的时序信号过程。最大模式下的时序除有些信号是由总线控制器(8288)产生的以外，其基本时间关系与最小模式大致相同。

8088 读-总线周期和 8088 写-总线周期分别如图 2-11 和图 2-12 所示。

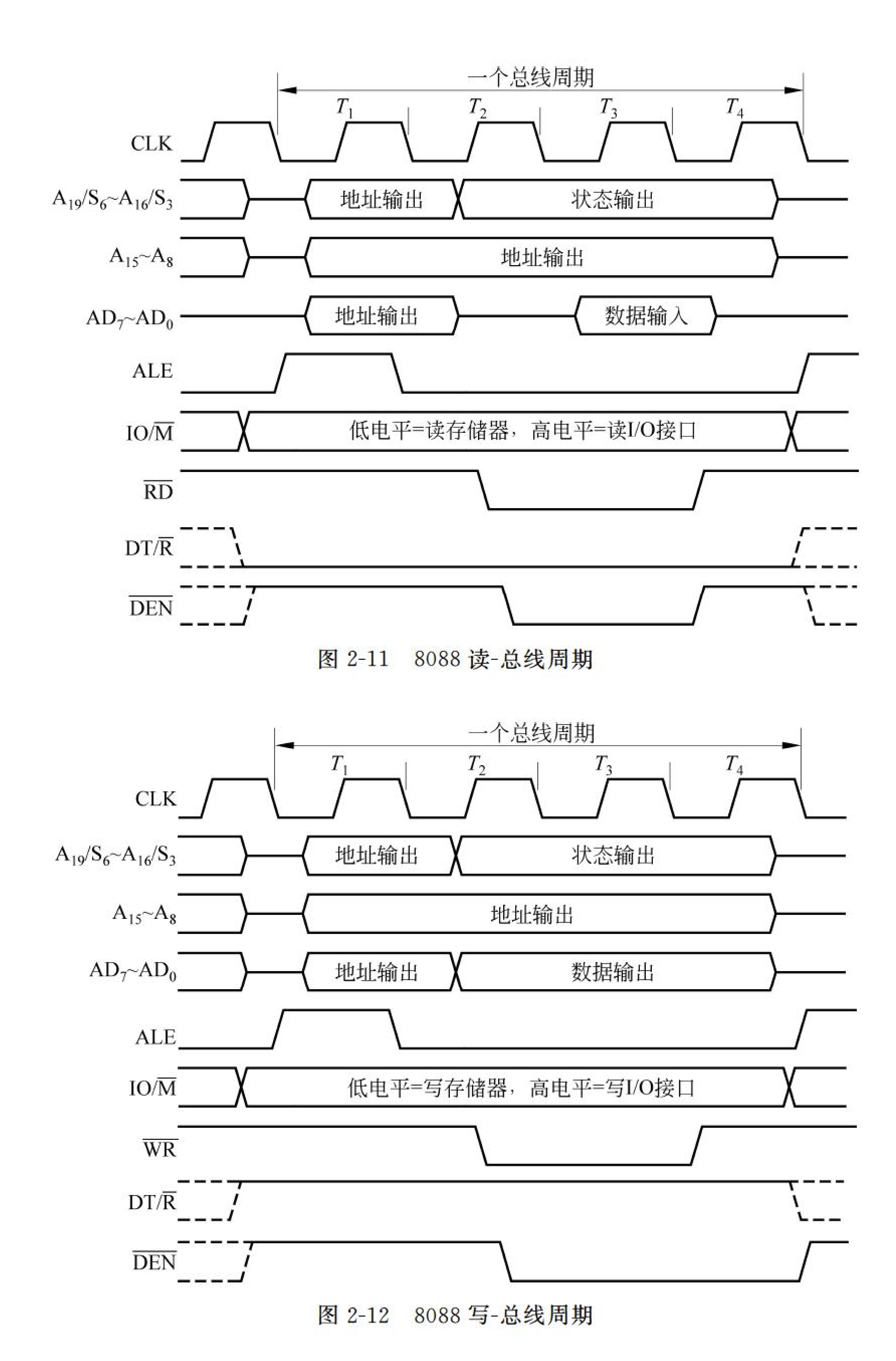


图 2-11　8088 读-总线周期

图 2-12　8088 写-总线周期

由图 2-11 和图 2-12 可知，正常的 8088 总线周期，不管是读或写，都由至少 4 个时钟周期（$T_1 \sim T_4$）组成。在 T_1 期间，地址信号线 $A_{15} \sim A_8$、地址/状态复用信号线 $A_{19}/S_6 \sim A_{16}/S_3$ 和地址/数据复用信号线 $AD_7 \sim AD_0$ 分别输出地址 $A_{15} \sim A_8$、$A_{19} \sim A_{16}$ 和 $A_7 \sim A_0$，同时输出地址锁存允许信号 ALE。外部电路利用 ALE 将地址信号锁存到地址锁存器中，即在锁存器输出端得到完整的 20 位地址信号。之后，就可利用 $IO/\overline{M}$、$\overline{RD}$、$\overline{WR}$等有关控制信号完成对内存或外设的读写操作。在写总线周期中，CPU 从 T_2 开始把数据

送到总线上并维持到 T_4。在读总线周期中,CPU 在 T_4 开始时刻读入总线上的数据。

如果内存或接口的速度比较慢,使得在 4 个时钟周期里不能完成读写操作时,可通过时钟产生器(8284)产生一个低电平信号送到 8088 的 READY 端。8088 CPU 在每个总线周期的 T_3 开始处都要检查 READY 的状态。若此时 READY 为低电平,则 CPU 不执行 T_4 而是在 T_3 之后插入一个等待时钟周期 T_w(图中未画出),以等待存储器或 I/O 接口完成读写操作。在 T_w 的开始时刻,CPU 还要检查 READY 状态,若仍为低电平,则再插入一个 T_w。此过程一直进行到某个 T_w 开始时,READY 已经变为高电平,这时下一个时钟周期就是总线周期的最后一个时钟周期 T_4。由此可见,利用 READY 信号,CPU 可以插入若干个 T_w,使总线周期延长,达到可靠读写内存和 I/O 接口的目的。

另外还要注意一点,CPU 的读($\overline{RD}$)或写($\overline{WR}$)是在 T_4 开始时刻(或$\overline{RD}$、$\overline{WR}$信号的后沿)进行的,这时数据线上的数据已经到达稳定状态,只有这样,利用 READY 插入 T_w 周期才有意义。

2.3 80386 微处理器

1985 年 10 月,Intel 公司推出了与 8088/8086/80286 相兼容的高性能的 32 位微处理器 80386,它是为满足高性能的应用领域与多用户、多任务操作系统的需要而设计的。它的发布标志着微处理器自此从 16 位迈入了 32 位时代。

2.3.1 80386 微处理器的主要特性

与上一代微处理器相比,80386 主要具有以下几个特性。

(1) 采用全 32 位结构,其内部寄存器、ALU 和操作是 32 位,数据线和地址线均为 32 位,故能寻址的物理空间为 2^{32}=4GB。

(2) 提供 32 位外部总线接口,最大数据传输率为 32MB/s,具有自动切换数据总线宽度的功能。CPU 读写数据的宽度可以在 32 位到 16 位之间自由进行切换。

(3) 具有片内集成的存储器管理部件 MMU,可支持虚拟存储和特权保护,虚拟存储器空间可达 64TB(2^{46}字节)。存储器按段组织,每段最长 4000MB,因此 64TB 虚拟存储空间允许每个任务可拥有多达 16 384 个段。存储保护机构采用四级特权层,可选择片内分页单元。内部具有多任务机构,能快速完成任务的切换。

(4) 具有 3 种工作方式:实地址方式、保护方式和虚拟 8086 方式。实地址方式和虚拟 8086 方式与 8086 相同,已有的 8088/8086 软件不加修改就能在 80386 的这两种方式下运行;保护方式可支持虚拟存储、保护和多任务,包括了 80286 的保护方式功能。

(5) 采用了比 8086 更先进的流水线结构,使其能高效、并行地完成取指、译码、执行和存储管理功能。它具有增强的指令预取队列,能预取指令并进行内部指令排队。取指和译码操作均由流水线承担,处理器执行指令不需等待。其指令队列从 8086 的 6 字节增加到 16 字节长。

2.3.2 80386 的内部结构

80386 内部结构如图 2-13 所示。它由三大部分组成：总线接口部件(BIU)、中央处理部件(CPU)和存储器管理部件(MMU)。

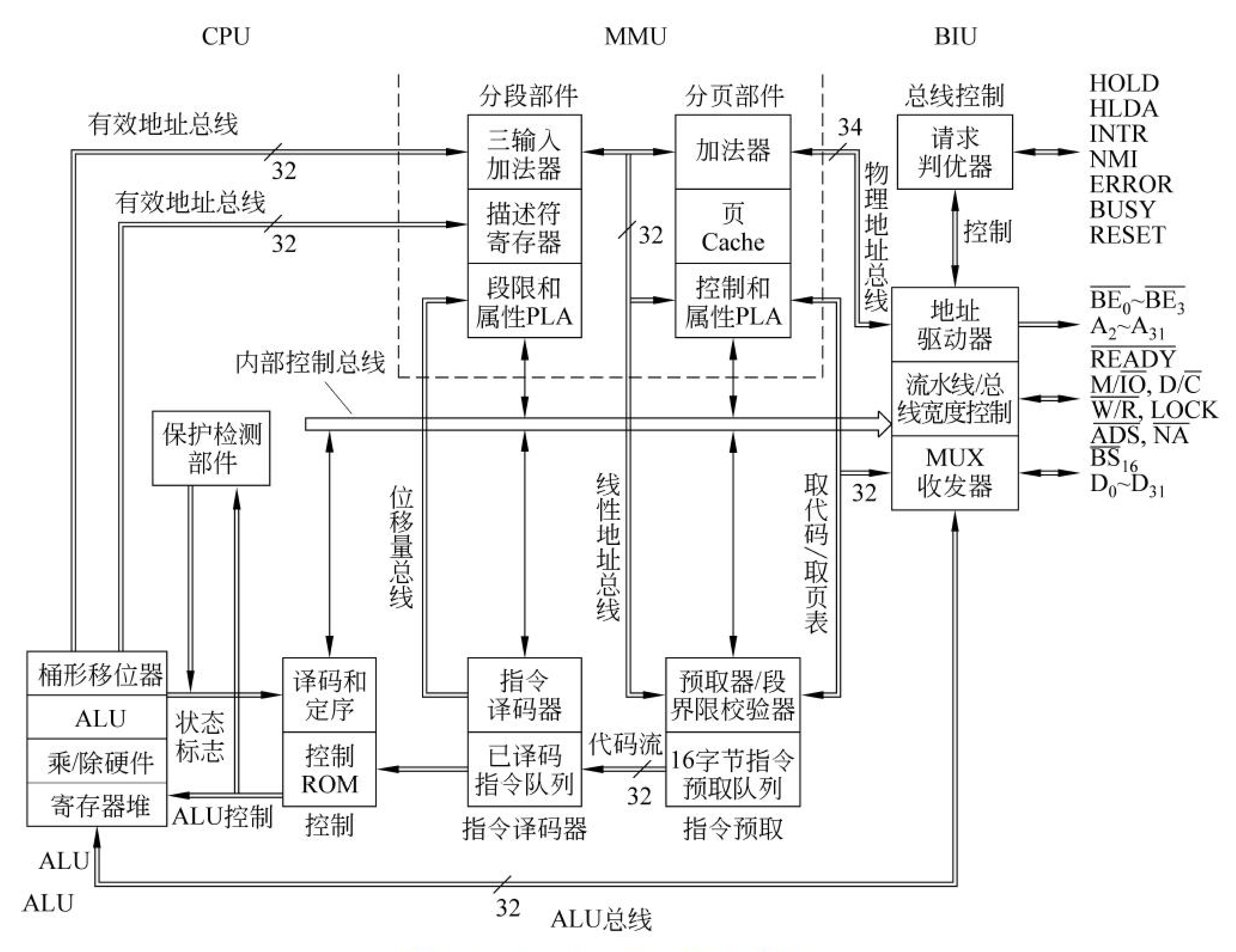


图 2-13 80386 微处理器内部结构

1. 总线接口部件

总线接口部件(Bus Interface Unit，BIU)负责与存储器和 I/O 接口进行数据传送，其功能是产生访问存储器和 I/O 端口所必需的地址、数据和命令信号。与 8088/8086 中的 BIU 作用相当。由于总线数据传送与总线地址形成可同时进行，所以 80386 的总线周期只有 2 个时钟。平常没有其他总线请求时，BIU 将下条指令自动送到指令预取队列。

2. 中央处理部件

中央处理部件(CPU)包括指令预取单元、指令译码单元和执行单元三部分。

1) 指令预取单元

指令预取单元(Instruction Prefetch Unit，IPU)负责从存储器中取出指令，放入

16 字节的指令队列中。它管理一个线性地址指针和一个段预取界限,负责段预取界限的检验,并将预取总线周期通过分页部件发给总线接口。每当预取队列不满或发生控制转移时,就向 BIU 发一个取指请求。指令预取的优先级别低于数据传送等总线操作。因此,绝大部分情况下是利用总线空闲时间预取指令。指令预取队列存放着从存储器取出的未经译码的指令。

2) 指令译码单元

指令译码单元(Instruction Decode Unit,IDU)负责从 IPU 中取出指令,进行译码,形成可执行指令,然后放入已译码指令队列以备执行部件执行。每当已译码指令队列中有空间时,就从预取队列中取出指令并译码。

3) 执行单元

执行单元(Execution Unit,EU)包括 8 个 32 位的寄存器组、1 个 32 位的算术逻辑单元 ALU、1 个 64 位桶形移位寄存器和 1 个乘法除法器。桶形移位器用来有效地实现移位、循环移位和位操作,被广泛地用于乘法及其他操作中。它可以在一个时钟周期内实现 64 位同时移位,也可对任何一种数据类型移任意位数。桶形移位器与 ALU 并行操作,可加速乘法、除法、位操作,移位和循环移位操作。

3. 存储器管理部件

存储器管理部件(Memory Management Unit,MMU)由分段部件和分页机构组成。

1) 分段部件

分段部件的作用是根据执行部件的请求,把逻辑地址转换成线性地址,在完成地址转换的同时还要执行总线周期的分段合法性检验。该部件可以实现任务之间的隔离,也可以实现指令和数据区的再定位。

2) 分页机构

分页机构的作用是把由分段部件或代码预取单元产生的线性地址转换成物理地址,并且要检验访问是否与页属性相符合。为了加快线性地址到物理地址的转换速度,80386 内设有一个页描述符高速缓冲存储器(TLB),其中可以存储 32 项页描述符,使得在地址转换期间大多数情况下不需要到内存中查页目录表和页表。试验证明,TLB 的命中率可达 98%。对于在 TLB 内没有命中的地址转换,80386 设有硬件查表功能,从而缓解了因查表引起的速度下降问题。

2.3.3 80386 的主要引脚信号

80386 共有 132 条引脚,使用 PGA 封装技术。它对外直接提供了独立的 32 位地址总线和 32 位数据总线,能在 2 个时钟周期内完成 32 位数据传送,在 33MHz 工作频率下,其传送速率为 66MB/s。其主要引脚信号如下。

(1) CLK_2:两倍时钟输入信号。该信号与 80384 时钟信号同步输入,在 80386 内部二分频后产生指令执行时钟 CLK。每个 CLK 由两个 CLK_2 时钟周期组成,分别称其为相 1 和相 2。

(2) $D_0 \sim D_{31}$：数据总线信号，双向三态。数据总线信号一次可传送8、16、32位数据。

(3) $A_2 \sim A_{31}$：地址总线信号输出，三态。与$\overline{BE_0} \sim \overline{BE_3}$相结合可起到32位地址的作用。

(4) $\overline{BE_0} \sim \overline{BE_3}$：字节选通输出信号。每条线控制选通一个字节，其状态根据内部地址信号A_0、A_1产生。$\overline{BE_0} \sim \overline{BE_3}$分别对应选通$D_0 \sim D_7$、$D_8 \sim D_{15}$、$D_{16} \sim D_{23}$与$D_{24} \sim D_{31}$，相当于存储器分为4个存储体，与$A_2 \sim A_{31}$结合可寻址$2^{30} \times 2^2 = 4G$个内存单元。

(5) $W/\overline{R}$：读/写控制，输出信号。

(6) $D/\overline{C}$：数据/控制输出信号。$D/\overline{C}$表示是数据传送周期还是控制周期。

(7) $M/\overline{IO}$：存储器与I/O选择信号，输出。

(8) LOCK：总线锁定输出信号。

(9) $\overline{ADS}$：地址状态，三态输出信号。$\overline{ADS}$表示总线周期中地址信号有效。

(10) $\overline{NA}$：下一地址请求，输入信号。$\overline{NA}$允许地址流水线操作，即当前周期发下一总线周期地址的状态信号。

(11) $\overline{BS_{16}}$：总线宽度为16的输入信号。

(12) $\overline{READY}$：准备就绪，输入信号。$\overline{READY}$表示当前总线周期已完成。

(13) HOLD：总线请求保持，输入。

(14) HLDA：总线响应保持，输出。

(15) PEREQ：处理器扩展请求，输入。PEREQ表示80387要求80386控制它们与存储器之间的信息传送。

(16) $\overline{BUSY}$：协处理器忙，输入。

(17) $\overline{ERROR}$：协处理器出错，输入。

(18) NMI：不可屏蔽中断请求信号，输入。

(19) INTR：可屏蔽中断请求信号，输入。

(20) RESET：复位信号。

2.3.4 80386的内部寄存器

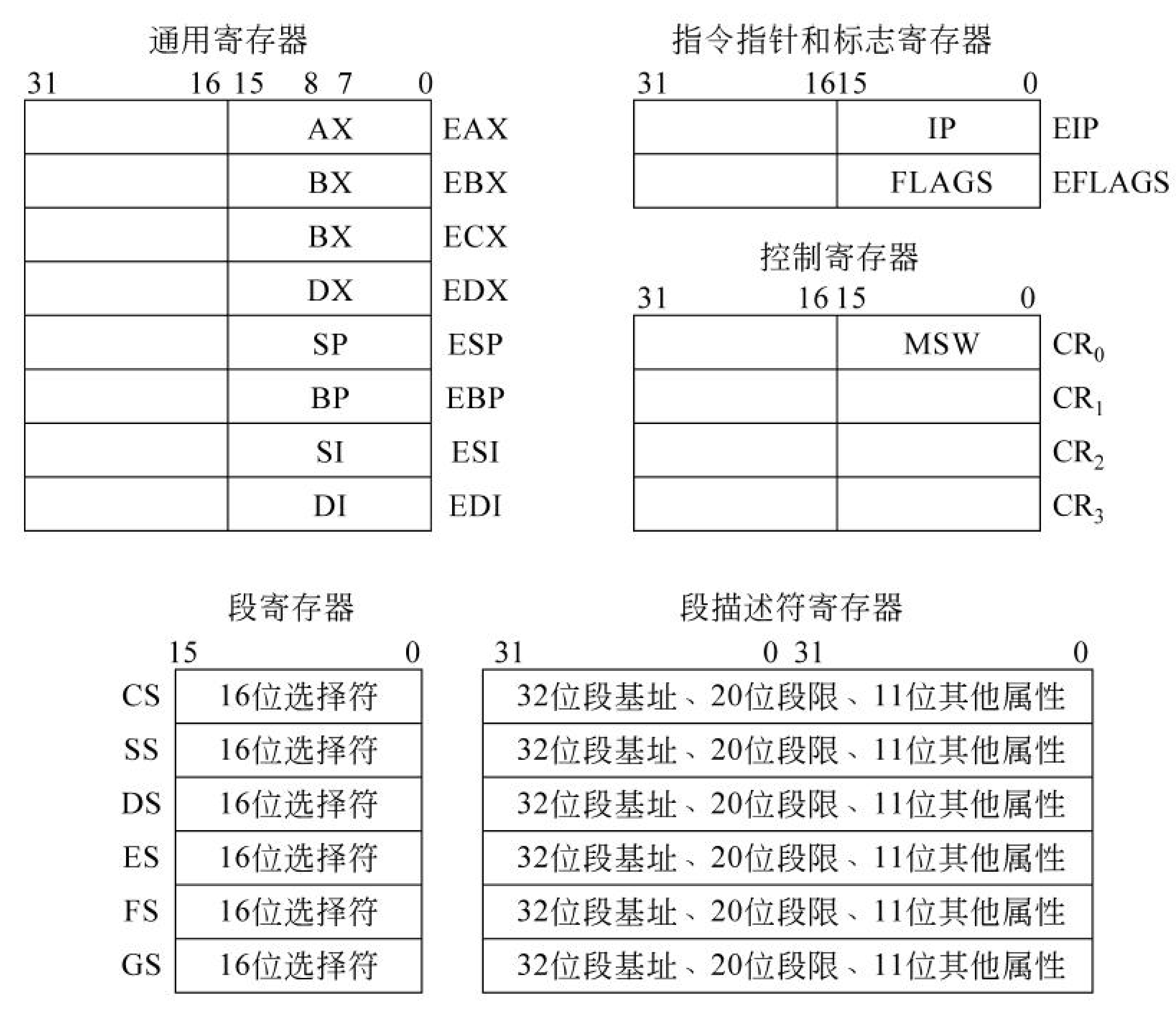
80386共有34个寄存器，可分为7类，如图2-14所示。它们分别是通用寄存器、指令指针和标志寄存器、段寄存器、系统地址寄存器、控制寄存器、调试寄存器和测试寄存器。

1. 通用寄存器

80386有8个32位的通用寄存器，都是由8088/8086相应的16位通用寄存器扩展而成的，分别是EAX、EBX、ECX、EDX、ESI、EDI、EBP、ESP。每个寄存器的低16位可单独使用，与8088/8086相应的16位通用寄存器作用相同。同时，AX、BX、CX、DX寄存器的高、低8位也可分别当作8位寄存器使用。

2. 指令指针和标志寄存器

指令指针EIP是一个32位寄存器，是从8086的IP扩充而来的。



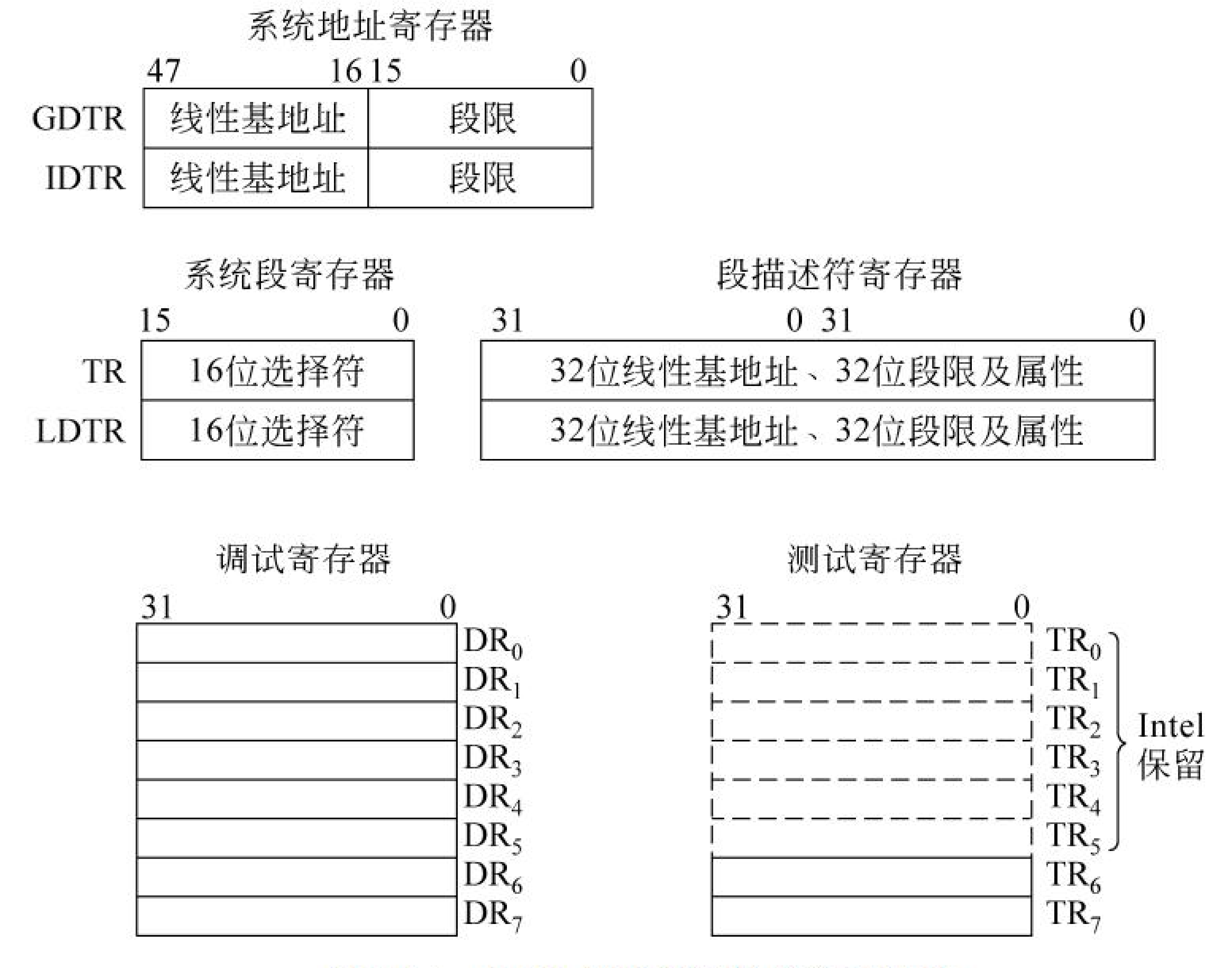


图 2-14 80386 微处理器的寄存器结构

80386 的标志寄存器 EFLAGS 也是一个 32 位寄存器，其中只使用了 15 位，如图 2-15 所示。32 位标志寄存器中，除保留 8088/8086 CPU 的 9 个标志外，另新增加了 4 个标志，其含义如下。

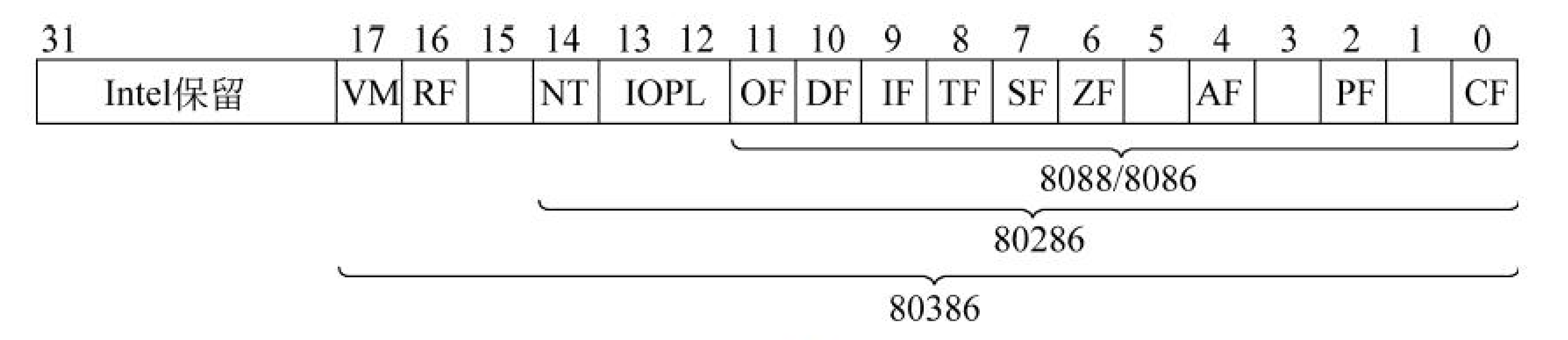


图 2-15 80386 的标志寄存器 EFALGS

(1) IOPL：I/O 特权级(I/O Privilege Level，位 13、12)。此标志位用以指定 I/O 操作处于 0～3 特权层中的哪一层。

(2) NT：嵌套任务(Nested Task，位 14)。若 NT=1，表示当前执行的任务嵌套于另一任务中，执行完该任务后，要返回到原来的任务中去；否则 NT=0。

(3) VM：虚拟 8086 方式(Virtual 8086 Mode，位 17)。若 VM=1，处理器工作于虚拟 8086 方式；若 VM=0，处理器工作于一般的保护方式。

(4) RF：恢复标志(Resume Flag，位 16)。RF 标志用于 DEBUG 调试，若 RF=0，调试故障被接受；RF=1，则遇到断点或调试故障时不产生异常中断。每执行完一条指令，RF 自动置 0。

3. 段寄存器

80386 有 6 个段寄存器，分别是 CS、DS、SS、ES、FS 和 GS。前 4 个段寄存器的名称与 8088/8086 相同，在实地址方式下的使用方式也和 8088/8086 相同。增加 FS 与 GS 主要是为了减轻对 DS 段和 ES 段的压力。

80386 内存单元的地址仍由段基地址和段内偏移地址组成。段内偏移地址为 32 位，由各种寻址方式确定。段基址也是 32 位，但除了在实地址方式下外，不能像 8086/8088 那样直接由 16 位段寄存器左移 4 位而得，而是根据段寄存器的内容，通过一定的转换得出。因此，为了描述每个段的性质，80386 内部的每一个段寄存器都对应着一个与之相联系的段描述符寄存器，用来描述一个段的段基地址、段界限和段的属性。每个段描述符寄存器有 64 位，其中 32 位为段基地址，另外 32 位为段限(本段的实际长度)和必要的属性。段描述符寄存器对程序员是不可见的。程序员通过 6 个段寄存器间接地对段描述符寄存器进行控制。在保护方式(多任务方式)下，6 个 16 位的段寄存器也称为段选择符，即段寄存器中存放的是某一个段的选择符。当用户将某一选择符装入一个段寄存器时，80386 中的硬件会自动用段寄存器中的值作为索引从段描述符表中取出一个 8 个字节的描述符，装入到与该段寄存器相应的 64 位描述符寄存器中。这个过程由 80386CPU 硬件自动完成。

一旦段描述符被装入段描述符寄存器中，在以后访问存储器时就可使用与所指定的段寄存器相应的段描述符寄存器中的段基地址来计算线性地址，而不必每次访问时都去查找段描述符表。

4. 控制寄存器

80386 有 4 个 32 位控制寄存器(CR_0、CR_1、CR_2 和 CR_3)，作用是保存全局性的机器

状态。其中 CR_0 的格式如图 2-16 所示。

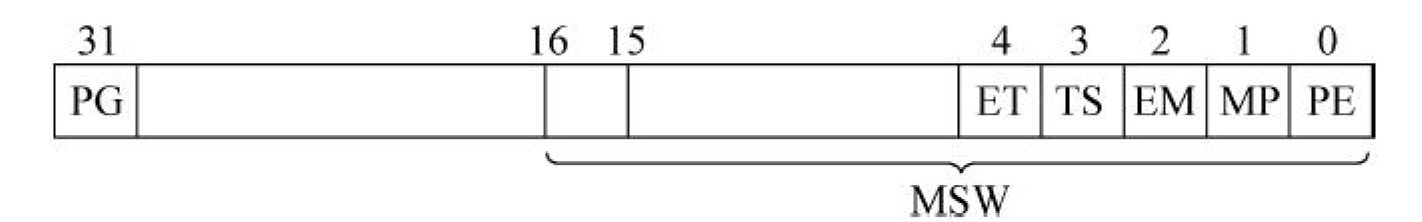


图 2-16 控制寄存器 CR_0 的结构

CR_0 的低 16 位称为机器状态字 MSW,其中主要几位如下。

(1) PE: 保护允许位。进入保护方式时 PE=1。除复位外,PE 位不能被清除。实方式时 PE=0。

(2) MP: 监视协处理器位。当协处理器工作时 MP=1,否则 MP=0。

(3) EM: 仿真协处理器位。当 MP=0,且 EM=1 时,表示要用软件来仿真协处理器功能。

(4) TS: 任务转换位。当两任务切换时,使 TS=1,此时不允许协处理器工作;当两任务之间切换完成后,TS=0。

(5) ET: 协处理器类型位。系统配接 80387 时 ET=1,配接 80287 时 ET=0。

(6) PG: 页式管理允许位。PG=1 表示启用芯片内部的页式管理系统,否则 PG=0。

CR_1 由 Intel 公司保留;CR_2 存放引起页故障的线性地址,只有当 CR_0 的 PG=1 时才使用 CR_2;CR_3 存放当前任务的页目录基地址,同样,仅当 CR_0 的 PG=1 时,才使用 CR_3。

5. 系统地址寄存器

系统地址寄存器有 4 个,用来存储操作系统需要的保护信息和地址转换表信息,定义目前正在执行任务的环境、地址空间和中断向量空间。

(1) GDTR: 48 位全局描述符表寄存器,用于保存全局描述符表的 32 位基地址和全局描述符表的 16 位界限(全局描述符表最大为 2^{16}字节,共 $2^{16}/8=8K$ 个全局描述符)。

(2) IDTR: 48 位中断描述符表寄存器,用于保存中断描述符表的 32 位基地址和中断描述符表的 16 位界限(中断描述符表最大为 2^{16}字节,共 $2^{16}/8=8K$ 个中断描述符)。

(3) LDTR: 16 位局部描述符表寄存器,用于保存局部描述符表的选择符。一旦 16 位的选择符放入 LDTR,CPU 会自动将选择符所指定的局部描述符装入 64 位的局部描述符寄存器中。

(4) TR: 16 位任务状态寄存器,用于保存任务状态段(TSS)的 16 位选择符。与 LDTR 相同,一旦 16 位的选择符放入 TR,CPU 会自动将该选择符所指定的任务描述符装入 64 位的任务描述符寄存器中。

LDTR 和 TR 寄存器由 16 位选择字段和 64 位描述符寄存器组成,用来指定局部描述符表和任务状态段 TSS 在物理存储器中的位置和大小。64 位描述符寄存器是自动装入的,程序员不可见。LDTR 与 TR 只能在保护方式下使用,程序只能访问 16 位选择符寄存器。

6. 调试寄存器

80386 设有 8 个 32 位调试寄存器 DR_0～DR_7，它们为调试提供了硬件支持。其中，DR_0～DR_3 是 4 个保存线性断点地址的寄存器；DR_4、DR_5 为备用寄存器；DR_6 为调试状态寄存器，通过该寄存器的内容可以检测异常，并进入异常处理程序或禁止进入异常处理程序；DR_7 为调试控制寄存器，用来规定断点字段的长度、断点访问类型、“允许”断点和“允许”所选择的调试条件。

7. 测试寄存器

80386 有 8 个 32 位的测试寄存器 TR_0～TR_7，其中 TR_0～TR_5 由 Intel 公司保留，用户只能访问 TR_6、TR_7。它们用于控制对 TLB 中的 RAM 和 CAM 相连存储器的测试。TR_6 是测试控制寄存器，TR_7 是测试状态寄存器，保存测试结果的状态。

2.3.5 80386 的工作模式

80386 可工作于实地址模式或保护虚地址模式。

1. 实地址模式

当 80386 加电或复位后，就进入实地址工作模式。80386 所有指令在实地址模式下都是有效的，不过操作数默认长度是 16 位，物理地址形成与 8088/8086 一样，将段寄存器内容左移 4 位与偏移地址相加而得到，寻址空间为 1MB，只有地址线 A_2～A_{19}，$\overline{BE_0}$～$\overline{BE_3}$ 是有效的。而 A_{20}～A_{31} 总是低电平，唯一的例外是在复位后，在执行第一条段间转移或调用指令前，所有访问代码段的总线周期的地址 A_{20}～A_{31} 输出总是高电平，以保证执行高端内存引导 ROM 中的指令。实地址模式下段的大小为 64KB，因此 32 位的有效地址必须小于 0000FFFFH。此模式下保留了两个固定的存储区域，它们是专用的。

(1) 中断向量表区：00000H～003FFH，在 1KB 存储空间保留 256 个中断服务程序的入口地址，每个入口地址占用 4 个字节，与 8088/8086 一样。

(2) 系统初始化区：FFFFFFF0H～FFFFFFFFH，存放 ROM 引导程序。

实地址模式下的物理地址变换如图 2-17 所示。

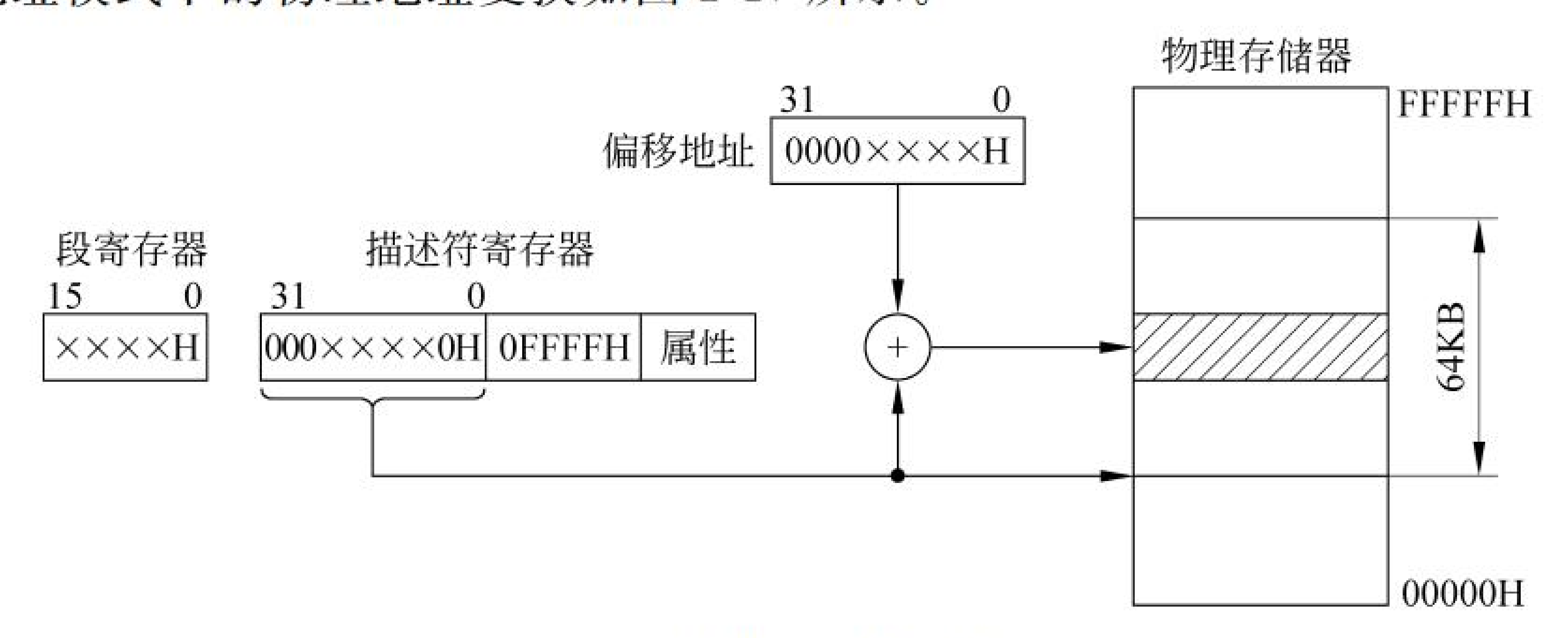


图 2-17　实模式下的地址变换

2. 保护虚地址模式

当 80386 工作在保护方式时，其能够访问的线性地址空间可达 4GB(2^{32})，而且允许运行几乎不受存储空间限制的虚拟存储器程序。用户逻辑地址空间(即虚拟地址空间)可达 64TB(2^{46})。在此模式下，80386 提供了复杂的存储管理和硬件辅助的保护机构，并可运行现有 8088/8086/80286 的所有软件。另外，它还增加了支持多任务操作系统的特别优化指令。

实际上，64TB 的虚拟地址空间是在磁盘等外部存储器的支持下实现的，它与 CPU 的存储器管理部件、48 位和 16 位的描述符表寄存器等有直接关系。编写程序时，程序可以放在磁盘存储器上，但在执行时，必须把程序加载到物理内存储器中。存储器管理就是要解决这个问题，要将 46 位虚拟地址变换成 32 位物理地址。

1) 保护模式的地址变换

在保护模式下，段寄存器中的内容不再是段的基地址，32 位的段基地址存放在一个段描述符表中，而段寄存器的内容作为选择符，即作为段描述符表的索引，用它来从表中取出相应的段描述符(包括 32 位段基地址、段界限和访问权等)。地址转换的过程是：由选择符的高 13 位作为偏移量，以 CPU 内部预先初始化好的 GDTR 的内容作为描述符表基地址，从表中获得相应的描述符，再将该描述符存入描述符寄存器。描述符中的段基地址(32 位)同指令给出的 32 位偏移地址相加得到线性地址，再通过分页机构进行变换，最后得到物理地址。如果不分页，线性地址就等于物理地址。保护模式下的地址变换如图 2-18 所示。

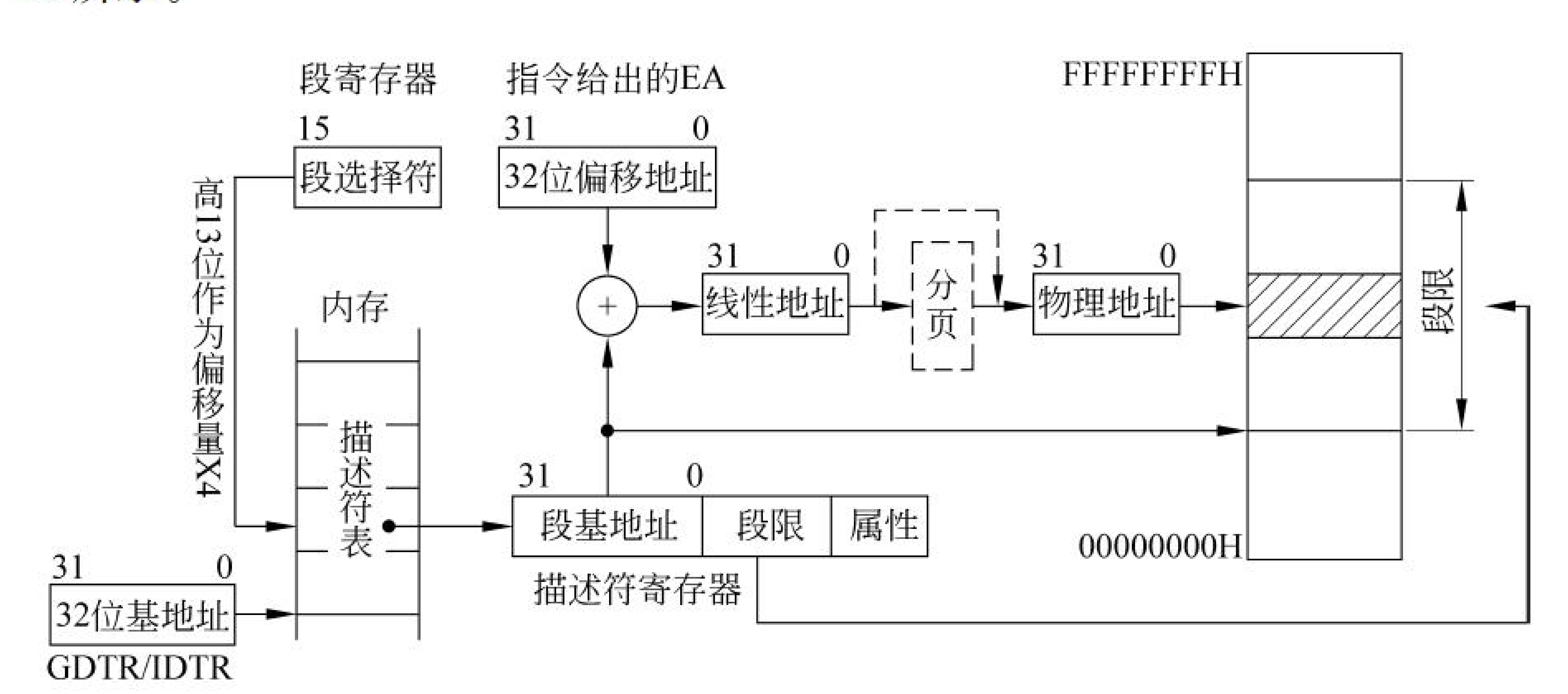


图 2-18 保护模式下的地址变换

2) 描述符表

(1) 段选择符。在保护虚地址模式下，段寄存器就成为一个选择符，由 3 个字段组成，如图 2-19 所示。

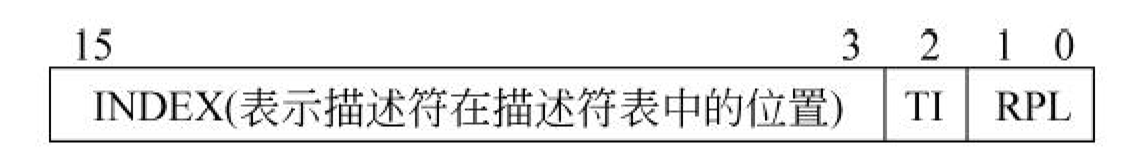


图 2-19 放在段寄存器中的选择符的格式

① INDEX：13 位索引值。INDEX 给出要选择的描述符在描述符表中的位置或描述符在表中的序号。INDEX 加上 32 位的描述符表基地址（在 GDTR/LDTR 中）就得到所选择描述符的物理地址。最大可选择 8K 个全局描述符和 8×2^{10} 个局部描述符，共 16×2^{10} 个描述符。

② TI：描述符表指示器。TI＝0，表示指向全局描述符表 GDT；TI＝1，表示指向局部描述符表 LDT。

③ RPL：选择器特权级。RPL 定义当前请求的特权层级别，特权级为 0～3 级，0 级最高，3 级最低。

（2）段描述符。用段选择器从描述符表中选择的对象称段描述符。段描述符包含了一个存储分段的所有信息，其中包括段的线性基地址和此段的界限（大小）及一些属性。如：此段的保护等级；读、写、执行特权和保护特权级别；操作数的默认长度（16 位或 32 位）；段的类型和段的粒度（即段的长度单位）；等等。

每个段描述符有 8 个字节，其中段基地址 32 位（4 个字节）、段界限 20 位，还有 12 位定义了段的属性信息，其格式如图 2-20 所示。

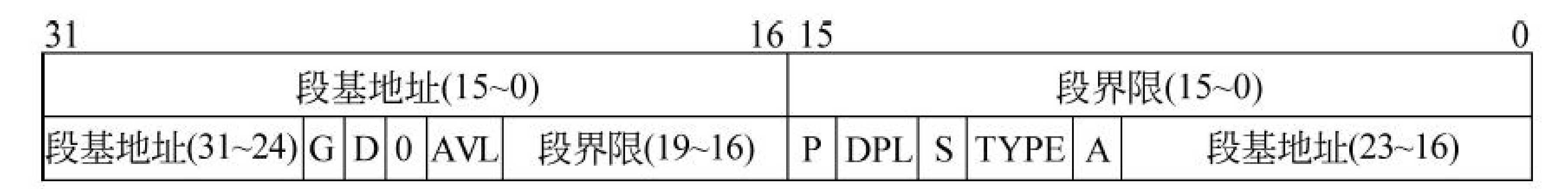


图 2-20　段描述符的格式

段描述符的 32 位基地址规定了存储分段在线性地址空间中的基地址。段界限指定了该存储分段的长度。粒度位 G＝0 时，段长以字节为单位，此时，该段的最大地址空间是 1MB；G＝1 时，段长以页为单位，此时，该段的最大地址空间是 4GB。D 位指示了操作数和有效地址的默认长度：D＝1，使用 32 位操作数和 32 位寻址方式；D＝0，使用 16 位操作数和 16 位寻址方式。P 位表示该存储分段是否在内存：P＝1，表示该分段已在内存；P＝0，表示该分段在外存硬盘交换区。在装入段寄存器时，若由段选择符寻址的段描述符的 P＝0，则产生类型 11 异常中断；若装入堆栈段寄存器，则产生类型 12 异常中断。系统响应异常事件后，操作系统将该分段从外存调入内存，置 P＝1，并且重新执行引起不存在异常事件的指令。

80386 把段分成两类：系统段和非系统段。系统段描述操作系统的系统表、任务和门的信息；非系统段就是代码和数据段。段描述符中的 S、TYPE 和 A 字段有 3 种组合方式：数据段或堆栈段（S＝1、TYPE 字段的 E＝0）；代码段（S＝1 且 TYPE 字段的 E＝1）；系统段（S＝0）。

2.4　Pentium 4 微处理器

与前一代微处理器相比，Pentium 处理器在结构上进行了一些新的改进，如增加了多媒体处理部件、集成了更大的缓存，将 L2 缓存也做到了 CPU 内部、采用了多级、多

流水线的超标量结构等。其主要的程序特性，如支持多用户、多任务，具有硬件保护功能，支持分段分页虚拟存储等均与 80386 CPU 类似。限于篇幅，这里仅简单介绍一下 Pentium CPU 家族中的 Pentium 4 的技术特点，其详细原理及相关理论可参阅其他有关书籍。

2.4.1 Pentium 4 微处理器中的新技术

1. 主要技术指标

Pentium 4 是一款基于 Intel NetBurst 微体系结构(Intel Micro-architecture)的微处理器。Intel NetBurst 微体系结构是一种允许处理器运行在极高时钟速率，比之前的 IA-32 结构具有更高性能级别的新型 32 位微体系结构。

采用 NetBurst 体系结构的奔腾 4 处理器具有以下先进的特性。

(1) 首次实现了 Intel NetBurst 微体系结构。

① 快速执行引擎使处理器的算术逻辑单元执行速度达到了内核频率的两倍，从而实现了更高的执行吞吐量。

② 超长流水线技术使流水线深度比 Pentium Ⅲ增加了一倍，达到 20 级，显著提高了处理器性能和执行速度。

③ 创新的新型高速缓存子系统使指令执行更加有效。

④ 增强的动态执行结构可以对更多的指令进行转移预测处理(比 Pentium Ⅲ处理器多 3 倍)，有效地避免因发生程序转移使流水线停顿的现象(因为一旦预测不正确，CPU 将不得不重新填充指令队列)。

(2) 流式 SIMD(单指令多数据)扩展 2(SSE2)技术。

① 扩展了 MMX(多媒体增强指令集)和 SSE(单指令多数据流式扩展)技术。新增加了 144 条多媒体处理指令，可用来支持：128 位 SIMD 整数运算和 128 位 SIMD 双精度浮点运算。

② 进一步增强和加速了视频、语音、数据加密、图像和影像处理。

(3) 400MHz 英特尔 NetBurst 微体系结构系统总线。

① 提供了 3.2GB/s 的吞吐率(比 Pentium Ⅲ快 3 倍)。

② 4 倍速的 100MHz 可升级的总线时钟使有效速度达到 400MHz。

③ 片段化事物，深度流水线化操作。

④ 128 字节传输管线，每次存取 64 字节。

(4) 与已有的为 IA-32 体系结构而编写的应用和操作系统完全兼容。

Pentium 4 的大小为 217 mm^2，内含 3400 万(新版本为 4200 万)个晶体管，采用 0.18μm 工艺制造，外部引脚数为 423 个。其外形封装如图 2-21 所示。

2. 流式 SIMD 扩展 2 技术

Pentium 4 处理器的流式 SIMD 扩展 2(SSE2)对 MMX 技术和 SSE 扩展进行了增

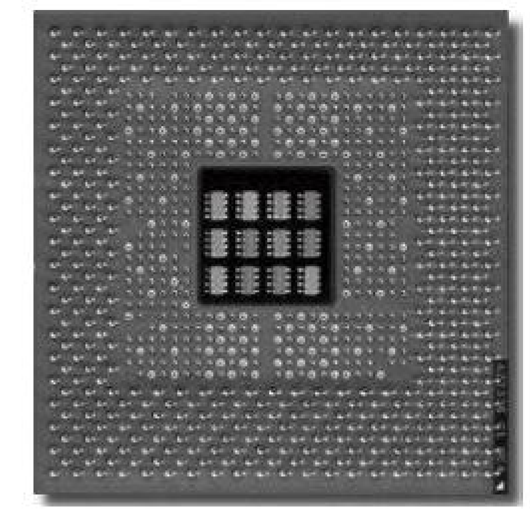



图 2-21　Pentium 4 微处理器的正面和背面

强,其中包括对紧缩型数据进行操作,使用 128 位宽的寄存器进行整数 SIMD 运算而增强了 SIMD 计算能力,新增加了对紧缩的双精度浮点数据类型和几种紧缩的 128 位整数类型的支持。这些新的数据类型允许紧缩的双精度、单精度浮点以及紧缩的整型数运算在 XMM 寄存器中进行。

新增加的 144 条 SIMD 指令包括浮点 SIMD 指令、整型 SIMD 指令、SIMD 浮点数和 SIMD 整型数互相转换的指令、在 XMM 和 MMX 寄存器之间的紧缩数据转换指令。新的浮点 SIMD 指令允许以紧缩的双精度浮点值进行运算(每个 XMM 寄存器存放 2 个双精度值)。SIMD 浮点的运算指令、单精度和双精度浮点格式均与二进制浮点算术运算的 IEEE 754 标准兼容。新的整型 SIMD 指令通过支持双字和四字的算术操作以及支持对紧缩的字节、字、双字、四字以及双四字的其他操作提供了灵活的极大动态范围的计算能力。SSE2 可同时使用以下类型的数据:4 个单精确浮点数、2 个双精确浮点数、16 个字节整数、8 个字整数、4 个双字整数、2 个四倍字整数、1 个 128 位长的整数。

丰富的数据类型和新增加的 SSE2 指令集大大提高了 Pentium 4 微处理器在多媒体应用领域的性能。

除了新的 128 位 SIMD 指令外,Pentium 4 也允许在 Pentium Ⅱ 和 Pentium Ⅲ 中就有的 68 条 SIMD 指令在 128 位的 XMM 寄存器中进行 128 位运算。这些增强的整型 SIMD 指令允许软件开发者开发具有更高性能的浮点和整数算法以及可以灵活地选用 XMM 寄存器或 MMX 寄存器编写 SIMD 代码。

为了加快处理速度以及增加高速缓存的利用率,SSE2 扩展提供了几条新的指令,以允许程序员来控制数据的可缓存能力。这些指令提供了使数据流入和流出寄存器而不破坏缓存的能力,还提供了在数据实际被使用前就预取的能力。

3. P6 系列的微体系结构

微体系结构是从 Pentium Pro 处理器开始被引入到 IA-32 处理器的,故通常将这种结构称为 P6 处理器微体系结构。P6 处理器微体系结构通过加入集成的 L2 缓存而使功能增强,这个 L2 缓存称为高级传输缓冲存储器(Advanced Transfer Cache)。这种微体系结构是一种 3 路超标量的流水线体系结构。3 路超标量是指使用并行处理技术的处理器平均每个时钟周期可执行 3 条指令的译码、分发和执行动作。为了控制这个指令流,P6 系列的处理器使用了一个非连接的 12 级超流水线来支持乱序(Out-of-Order)指令执行。图 2-22 为具有高级传输缓存的 P6 微体系结构流水线的概念框图。

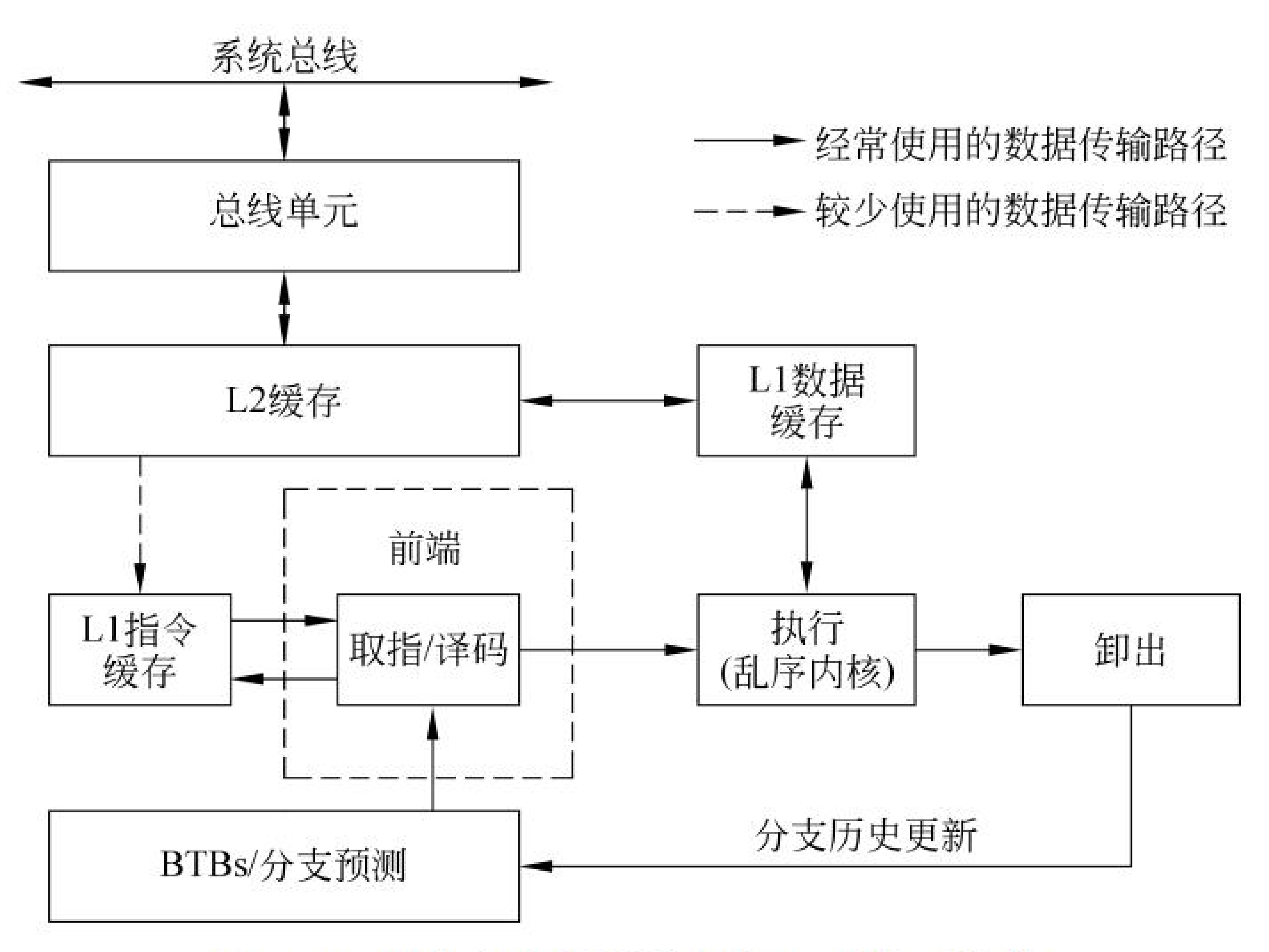


图 2-22　具有高级传输缓存的 P6 微体系结构

微体系结构的流水线分为 4 个部分(L1 和 L2 缓存、前端、乱序执行内核以及卸出)。指令和数据通过总线单元馈送到这些部件。

为了保证稳定地为执行指令流水线供应指令和数据,P6 微体系结构配置了两级缓存。第一级缓存提供 8KB 的指令缓存和 8KB 的数据缓存,它们都与流水线紧密地结合在一起;第二级缓存为 256KB、512KB 或 1MB 的静态 RAM,它通过一个全速的 64 位缓存总线与处理器内核相连。

P6 微体系结构的核心就是创新的乱序执行机制,称为动态执行(Dynamic Execution)。动态执行包括了 3 个数据处理的概念:深度分支预测(Deep Branch Prediction)、动态数据流分析(Dynamic Data Flow Analysis)、推测执行(Speculative Execution)。

深度分支预测是一种能够提供高性能流水线微结构的现代技术。它允许处理器在分支之前对可能要执行的指令译码以保持指令流水线满负荷运行。P6 处理器系列实现了高度优化的分支预测算法,通过多级分支、过程调用和返回来预测指令流的方向。

动态数据流分析包括通过处理器对数据流的实时分析来决定数据与寄存器的依赖关系并检测乱序指令执行的时机。

乱序执行核心能够同时监视许多指令并以使处理器多个执行单元的使用达到最优化的顺序来执行这些指令。乱序执行使处理器的执行单元保持忙碌,甚至当缓存未命中或指令之间出现了数据依赖关系也是如此。

推测执行是指处理器在条件转移的目标未知时就提前执行那些仅在条件转移发生后才能够确定执行的指令的能力,并且最终能够以原始指令流相同的顺序提交结果。为使推测执行成为可能,P6 处理器的微体系结构把指令的调度和执行与最终结果的提交分离开来。处理器的乱序执行核心利用数据流分析技术预先执行指令池中所有可能要被执行的指令,并把不同的执行结果存在临时寄存器中。接着,卸出单元顺序地搜索指令池,查找那些已正式被执行完的、与其他指令或尚无结果的转移预测不再有数据依赖关系的指令。若找到这样的指令,卸出单元就把这些指令的执行结果按原来指令的顺序提交到存

储器或寄存器，然后把这些指令从指令池中卸出。

通过分支预测、动态数据流分析和推测执行三者的有机结合，P6 微体系结构的动态执行能力消除了指令执行时在传统的取指和执行阶段之间其指令序列必须是线性顺序的限制。这样就能使处理器不间断地进行指令译码（甚至当指令流中有多级分支跳转时也是如此）；分支预测和先进的译码器能保证指令流水线始终处于满的状态。乱序推测执行引擎能够利用处理器的 6 个执行单元来并行地执行指令。最后，指令的结果能够按照原来程序中指令的顺序被提交以保证数据的完整性和程序的一致性。

4. Intel NetBurst 微体系结构

Intel NetBurst 微体系结构的概念框图如图 2-23 所示。它包括 3 个主要组成部分：有序执行的前端（Front End）流水线、乱序推测执行的内核、有序的指令流卸出部件。

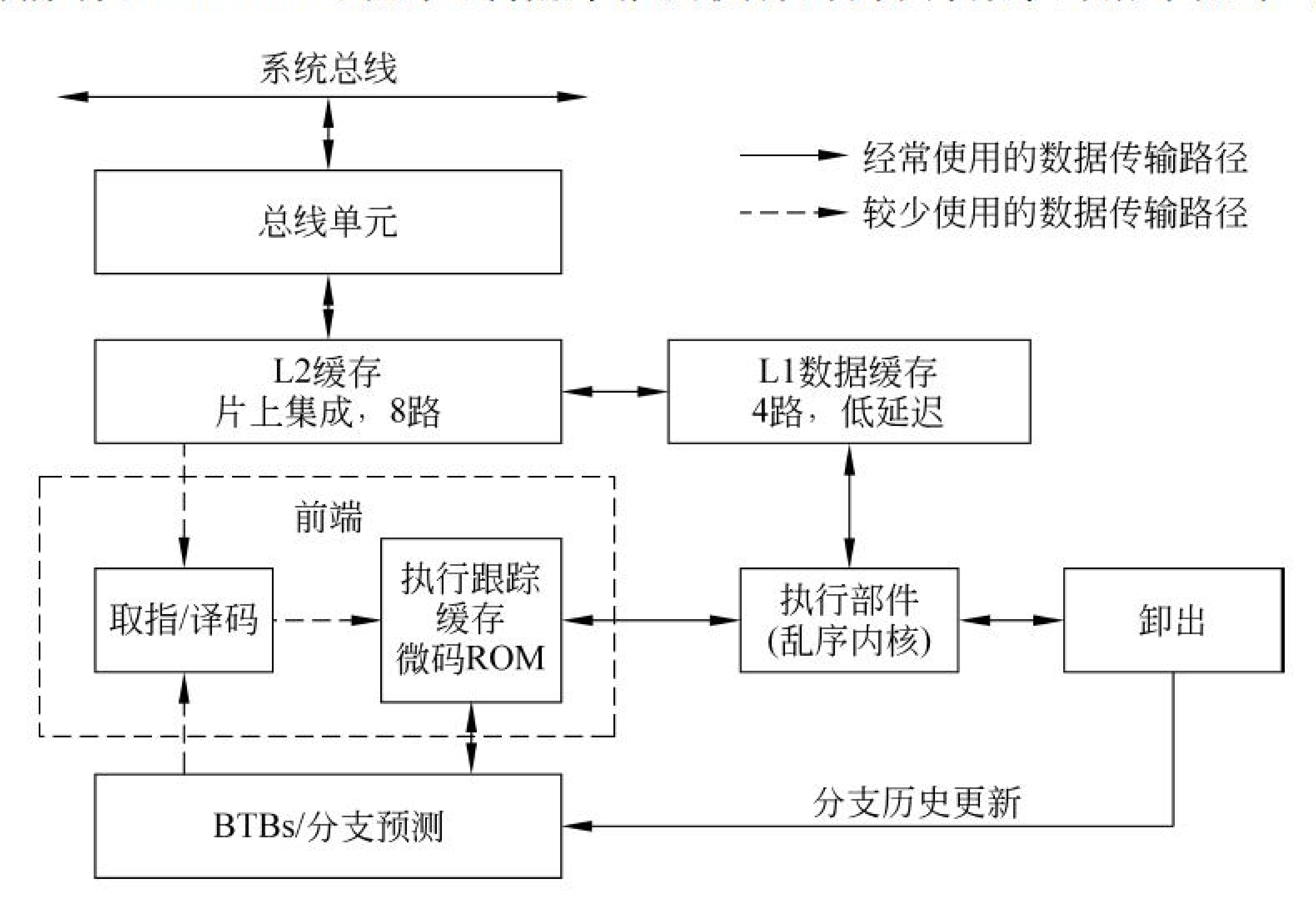


图 2-23　Intel NetBurst 微体系结构

1）前端流水线

前端的作用是按照程序原来的顺序为具有极高运行速度并能以 1/2 个时钟周期的延迟执行基本整型运算的乱序执行内核提供指令。前端执行取指操作并对指令译码，然后把它们分解为简单的微操作。前端能在一个时钟周期内以程序原来的顺序向乱序执行内核发出多个微操作。

前端完成以下几个基本的功能。

（1）预取那些可能要被执行的指令。

（2）取出未被预取的指令。

（3）把指令译码成微操作。

（4）为复杂指令和特殊指令生成微码。

（5）执行跟踪缓存送出译码后的指令。

（6）使用先进的预测算法来预测可能的程序分支。

Intel NetBurst 微体系结构的前端在设计时就考虑到了一些在高速流水线微处理器

中常见的问题。其中两个对延迟影响最大的问题是指令译码时间及由于分支或分支目标位于缓存流水线的中间而造成的译码时间的浪费。

为了解决这两个问题，Pentium 4 中取消了 L1 指令缓存，而代之以执行跟踪缓存。把已译码的指令保存在执行跟踪缓存中。Intel 公司的设计人员认为，在原来的 P6 微体系结构中，L1 指令高速缓存中的指令直到真正要被处理单元执行时才会取出进行译码，这样对某些复杂的 x86 指令需耗费太多的时间进行指令译码，以至于将拖延整个流水线执行的运作。另外在循环程序中，一段 x86 指令会被循环地多次执行，这样就使得每当这些指令进入执行路径一次就不得不再进行一次译码。此外，程序中的分支跳转预测错误时，L1 缓存也必须重新填充，这是 L1 指令高速缓存难以处理的问题。使用了执行跟踪缓存后，当重复执行某些指令时，就可从执行跟踪缓存取出译码后的指令直接执行，从而节省了这些指令的译码时间，避免了流水线的延迟。最重要的是，当超长流水线执行中出现分支预测错误时，流水线能及时从执行追踪缓存中快速地重新取得发生错误前已经过译码的指令，从而加速流水线填充过程。执行追踪缓存每两个时钟周期为流水线提供 6 个微指令，也就是每时钟周期提供 3 个微指令。追踪高速缓存大小为 96KB，一条 Pentium 4 的微指令长度为 64b 左右，所以追踪高速缓存可容纳约 12000 条微指令。

前端的执行过程是，首先由译码引擎取出指令并将其译码，然后经由微指令排序器(Micro Instruction Sequencer)将其序列化成一系列的微操作——称为轨迹(Traces)。这些微操作轨迹被存放在跟踪缓存中。一个分支指令所要转移到的可能性最大的目标轨迹紧跟在分支指令的轨迹后面而不管实际的分支指令下面的一条指令是什么。一旦轨迹被建立，就在跟踪缓存中查找跟在这个轨迹后面的那条指令。如果该指令是已存在轨迹中的第一条指令，从存储器中取指并进行译码的操作就会停止，跟踪缓存就成为下一条指令的来源地。Intel NetBurst 微体系结构中关键的执行循环如图 2-23 所示，它比图 2-22 所示的 P6 微体系结构中的关键执行循环要简单。

2）乱序执行内核

乱序执行能力是并行处理的关键所在。乱序执行使得处理器能够重新对指令排序，这样当一个微操作由于等待数据或竞争执行资源而被延迟时，后面的其他微操作也仍然可以绕过它继续执行。处理器拥有若干个缓冲区来平滑微操作流。这意味着当流水线的一个部分产生了延迟，该延迟也能够通过其他并行的操作予以克服或通过执行已进入缓冲区中排队的微操作来克服。

乱序执行内核按并行执行的要求来进行设计。它能在一个周期中发出 6 个微操作，这大大超过跟踪缓存和卸出部件执行微操作的速率。大多数流水线能够在每一个周期启动执行一个新的微操作，所以每条流水线能够允许一次穿越过多条指令。

3）指令卸出

卸出部分接收执行核心的微操作执行结果并处理它们，以便根据原始的程序顺序来更新相应的程序执行状态。为了保证执行在语义上正确，指令的执行结果在卸出前必须按照原始程序的顺序进行提交。

当一个微操作执行完成并把结果写入目标后，它就被卸出。每一周期被卸出的微操作多达 3 个。处理器中的重排序缓冲(Reorder Buffer，ROB)就是实现此功能的部件，它

缓冲执行结束的微操作、按原始顺序更新执行状态、管理异常的排序。

卸出部分还跟踪分支的执行并把更新了的转移目标送到 BTB 以更新分支历史。这样，不再需要的轨迹被清除出跟踪缓存，并根据更新过的分支历史信息来取出新的分支路径。

2.4.2 Pentium 4 CPU 的结构

1. Pentium 4 的功能结构

Pentium 4 的功能框图如图 2-24 所示。

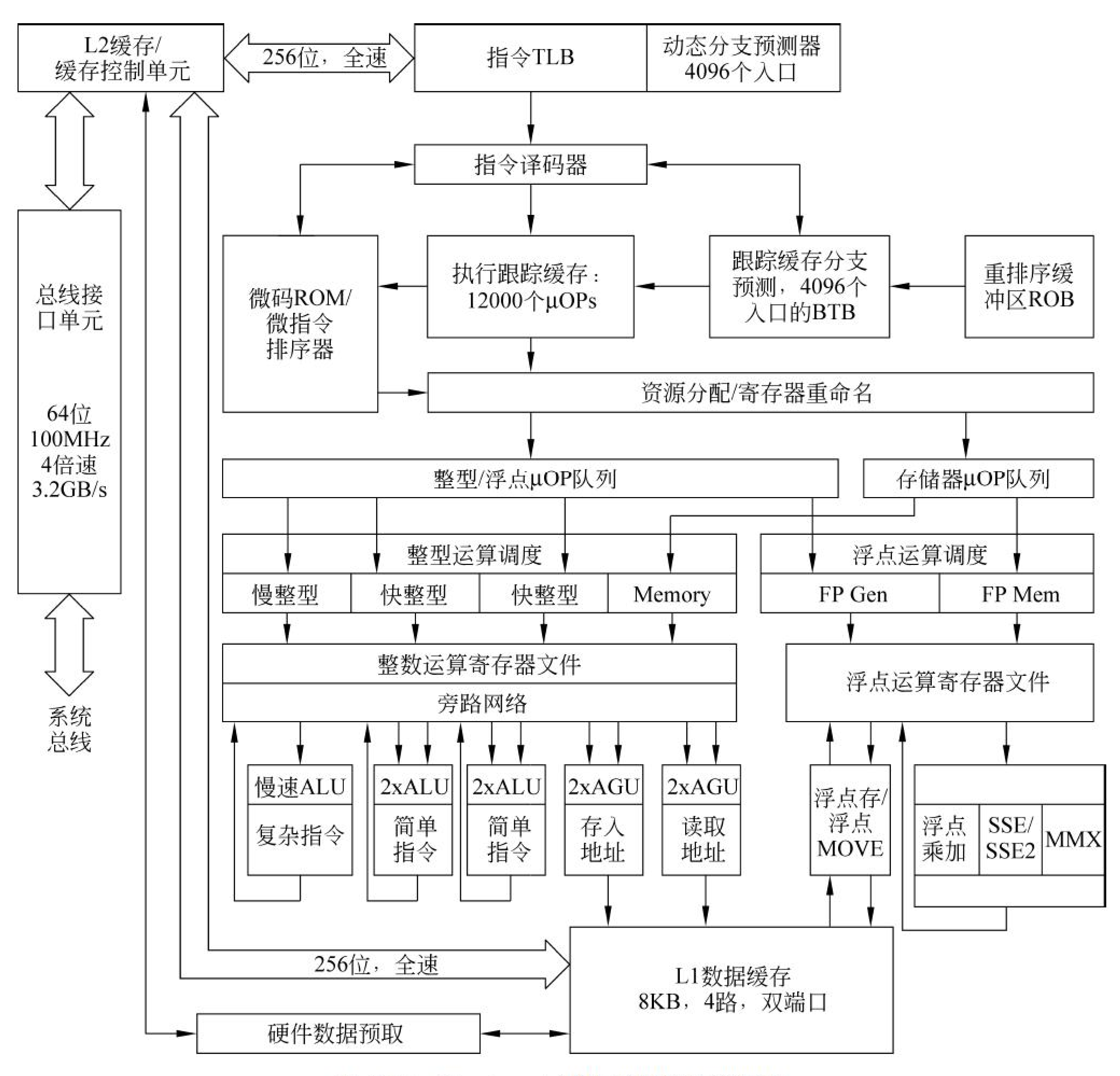


图 2-24 Pentium 4 微处理器的功能框图

(1) BTB：分支目标缓冲区(Branch Target Buffer)，用于存放所预测分支的所有可能的目的地址。

(2) μOP：微操作(Micro-Operation)。微处理器的执行部件能直接执行的指令称为

微指令(或微命令),执行部件接受微指令后所进行的操作称为微操作。一条指令可分解为一系列微指令的执行。与 x86 的变长指令集不同,微指令的长度是固定的,因此很容易在流水线中进行处理。指令译码部件会将 x86 指令转成一条或多条微指令,对于复杂的 x86 指令则会生成更多条微指令。例如,大多数 x86 指令会被编译成 1~2 条微指令,一些很简单的指令如 AND、OR、XOR、ADD 等仅被编译成 1 条微指令,而 DIV 、MUL 以及间接寻址运算则会生成较多的微指令,极为复杂的指令如三角函数等则可能会生成上百条微指令。在现代超标量微处理器中,微指令存放在内部的一个微码存储器(Micro-Code ROM)中。

(3) ALU: 算术逻辑单元,即整数运算单元。数学运算如加、减、乘、除以及逻辑及移位运算如 AND、OR、ASL、ROL 等指令都在算术逻辑单元中执行。在程序中,这些运算占了绝大多数,所以 ALU 的性能在很大程度上决定了系统的性能。

(4) AGU: 地址产生单元(Address Generation Unit)。AGU 负责在信息取出或存入时决定正确的地址。一般程序很少用绝对寻址的方式,程序中进行数组操作时,通常用的是间接寻址,这会使 AGU 单元持续处于忙碌状态。

2. Pentium 4 的系统结构

自从 80386 推出以来一直到现在最新推出的 Pentium 4,Intel 公司的 CPU 体系结构基本没有什么大的变化,Intel 公司将这一类 CPU 的体系结构统称为 IA-32 结构。图 2-25 对 IA-32 结构中的系统寄存器和数据结构进行了简要总结。

2.4.3 Pentium 4 的存储器管理

Pentium 4 继承了 IA-32 结构,所以它的存储器管理与 80386 基本相同,也包括了分段管理和分页管理。分段提供了隔离代码、数据和堆栈的机制,使多个程序(或任务)能够运行在同一个 CPU 上而不会互相干扰;分页提供了实现传统的基于页请求的虚存系统,这种系统当需要时能把程序执行环境的片段映射到物理存储器中。分页也能用于多个任务的隔离。当运行在保护方式时,必须使用某种形式的分段机制。分段机制不能通过状态位被禁止掉,而分页则是可选的。

如图 2-26 所示,分段把处理器的可寻址存储空间(称为线性地址空间)分成较小的、受保护的地址空间,称为段。段可用来存放代码、数据和堆栈或者存放系统数据结构(如 TSS 或 LDT)。当多个程序运行在同一个处理器上时,每一个程序都能够指定自己的段集合。处理器将限制这些段的界限,以保证一个程序不会把数据写到其他程序的段中干扰其他程序的运行。分段机制也允许指定段类型,以限制对某些特殊段的操作。

所有段都包含在处理器的线性地址空间中。为了定位某个段中的一个字节,则必须提供该字节的逻辑地址(又称为远指针)。正如已经知道的,逻辑地址由段选择符和偏移量两部分构成。段选择符是段的唯一标识,它提供了访问段描述符表(如 GDT)的偏移地址(或索引)。段描述符表中存放的是称为段描述符的数据结构。每一个段都有一个段描述符,段描述符定义了段的大小、访问权限和特权级、段的类型以及该段第一个字节在线

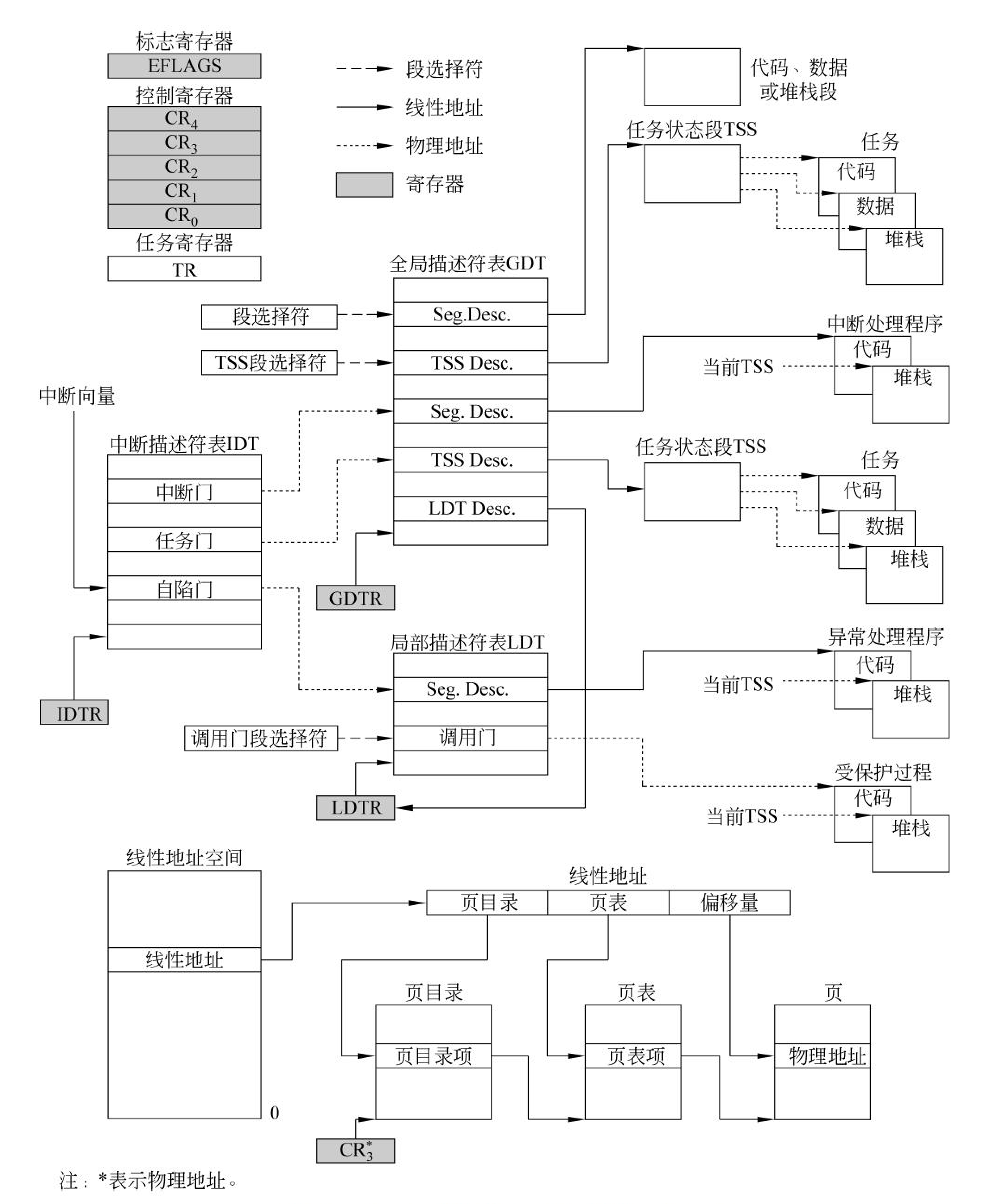


图 2-25　IA-32 结构中的系统寄存器和数据结构

性地址空间中的位置(称为段的基地址)。逻辑地址中的偏移量与段基地址相加就可以定位段中的任意字节。段基址加偏移量得到的值称为线性地址。若未使用分页,处理器的线性地址空间直接映射为物理地址空间。物理地址空间定义为处理器在它的地址总线上所能产生的地址范围。

由于多任务系统通常定义了一个比其拥有的物理存储器大得多的地址空间,这就需

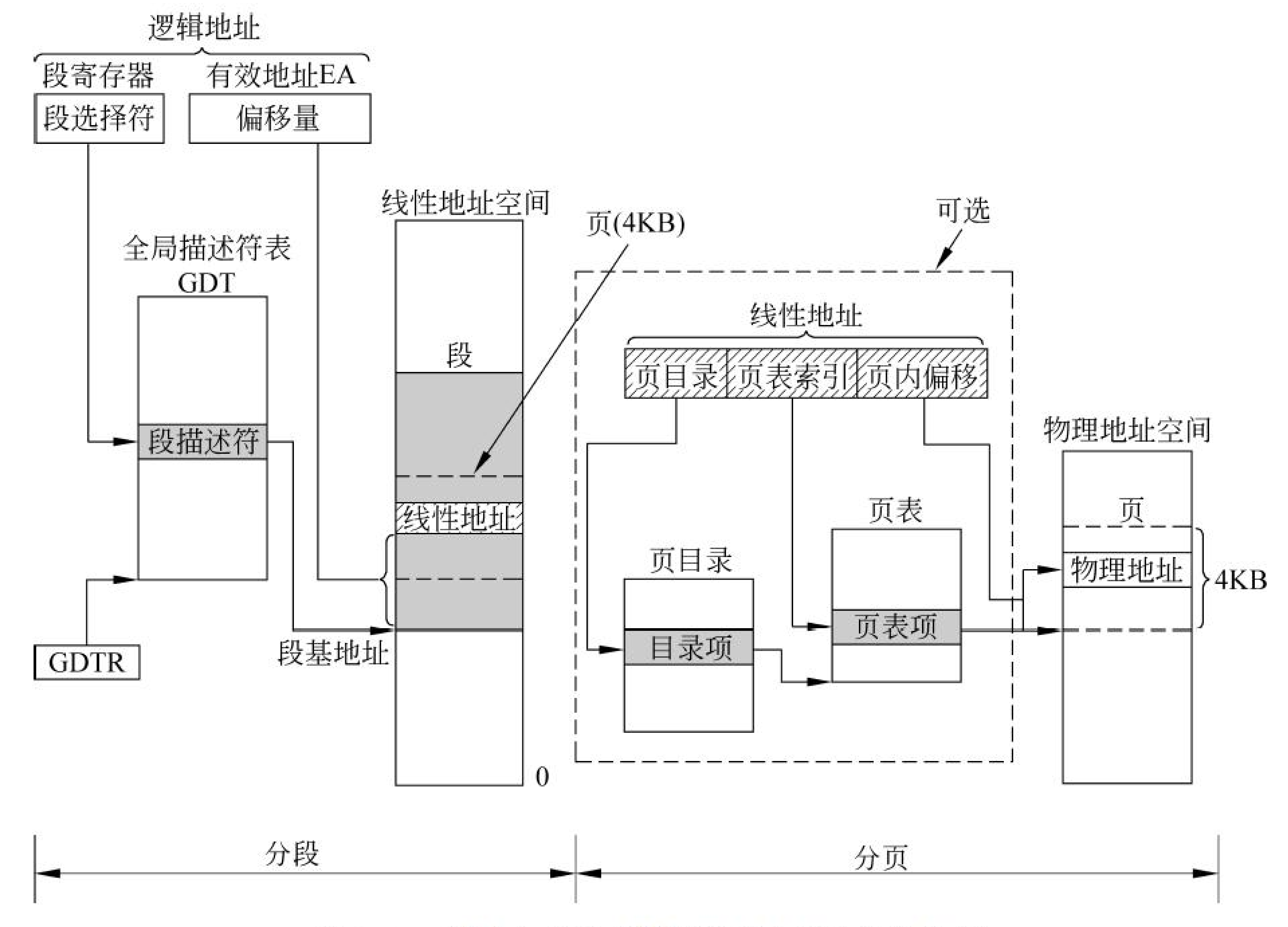


图 2-26 保护方式下存储器管理中的分段与分页

要有一个将线性地址空间虚拟化的方法。线性地址空间虚拟化是通过分页来实现的。

分页支持虚拟存储器环境，在这种环境中，用一个小容量的物理存储器(RAM 和 ROM)和一些磁盘空间来模拟一个非常大的线性地址空间。当使用分页时，每一个段都分为多个页(页面大小通常为 4KB)，页可存储在内存中或磁盘中。操作系统负责维护页目录和一个页表集合，以跟踪页的使用。当一个程序(或任务)要访问线性地址空间中的一个地址位置时，处理器使用页目录和页表把线性地址转换为存储器的物理地址，然后即可执行所请求的动作(读或写)。如果所访问的页不在物理存储器中，处理器就会暂时中断该程序的执行(通过产生一个页错误异常)，由操作系统把所需的页从磁盘读入物理内存中，然后接着执行由于页错误而被中断的程序。在物理内存和磁盘之间的页交换对应用程序来说是透明的。当处理器运行在虚拟 8086 模式时，为 16 位 CPU 编写的程序也可以被分页。

2.4.4 Pentium 4 的基本执行环境

Pentium 4 仍然继续支持 IA-32 结构的 3 种操作模式：保护模式、实模式和系统管理模式。Pentium 4 的基本执行环境对这 3 种模式来说都是相同的，环境包括以下可使用的资源，如图 2-27 所示。

(1) 地址空间。任何程序或任务都可以访问最大为 4GB(2^{32} 字节)的线性地址空间和最大为 64GB(2^{36} 字节)的物理地址空间。

(2) 基本的程序执行寄存器。基本的程序执行寄存器包括 8 个通用寄存器、6 个段寄

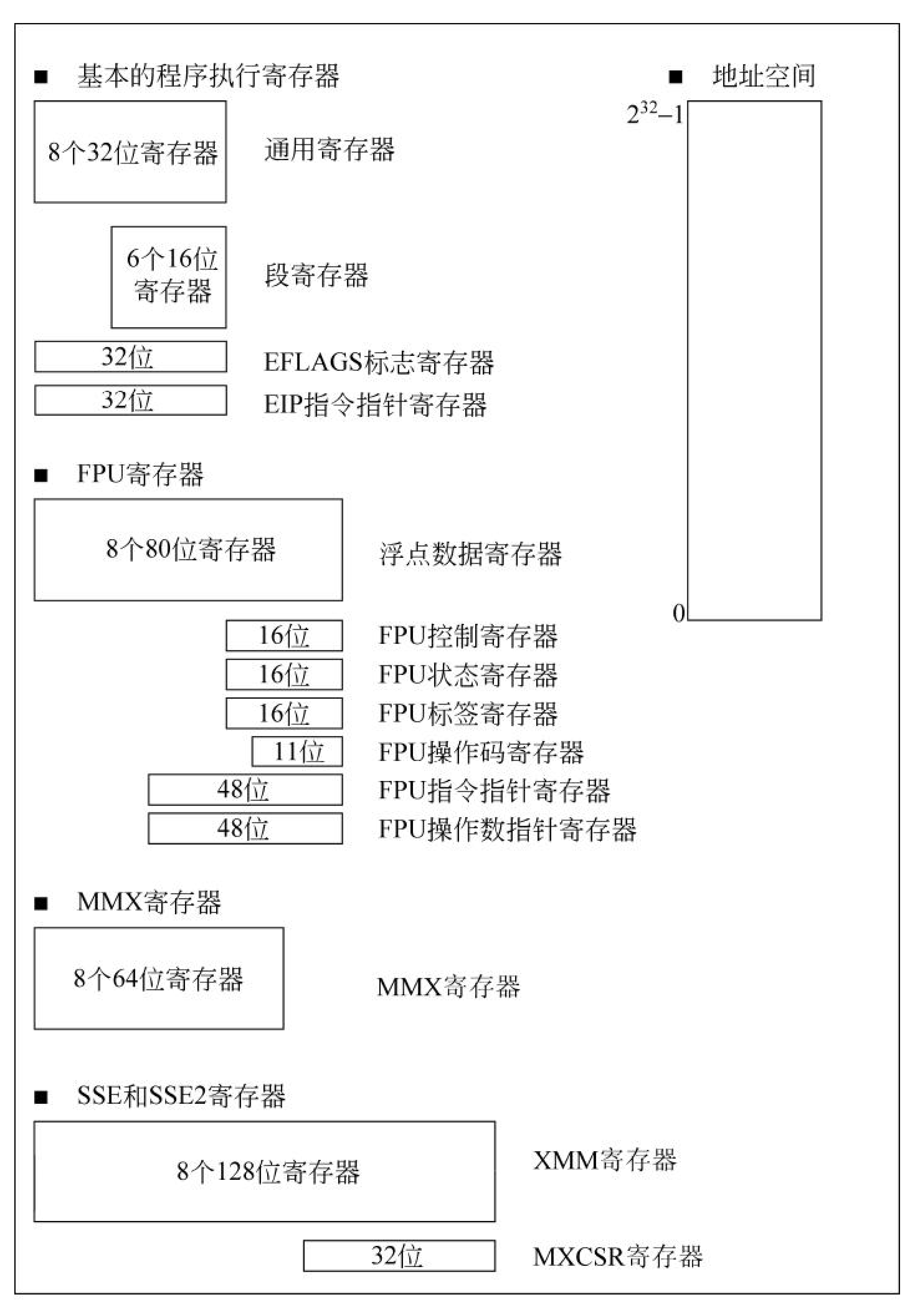


图 2-27　Pentium 4 的基本程序执行环境

存器、标志寄存器和指令指针寄存器。这些寄存器能够以字节、字和双字来执行基本的整型算术逻辑运算、进行程序流控制、进行位和字符串的操作以及访问存储器等。

(3) FPU 寄存器。FPU 寄存器包括 8 个 FPU 浮点数据寄存器、FPU 控制寄存器、FPU 状态寄存器、FPU 指令指针寄存器、FPU 操作数指针寄存器、FPU 标签寄存器和 FPU 操作码寄存器。这些寄存器用于单精度浮点数、双精度浮点数、扩充的双精度浮点数、字、双字、四字整型、BCD 数的运算。

(4) MMX 寄存器。8 个 MMX 寄存器用于执行单指令多数据(SIMD)操作,支持 64 位紧缩的字节、字和双字整数类型。

(5) SSE 和 SSE2 寄存器。SSE 和 SSE2 包括 8 个 XMM 寄存器和 1 个 MXCSR 寄存器,支持 128 位紧缩的单精度浮点数、双精度浮点数以及 128 位紧缩的字节、字、双字、四字整型数的 SIMD 操作。

(6) * 堆栈。堆栈用于支持过程(子程序)调用和向过程传递参数。

(7) * I/O 端口。

(8) * 控制寄存器。5 个控制寄存器(CR_0～CR_4)决定了处理器的操作模式和当前任务的特征。

(9) * 存储管理寄存器。GDTR、IDTR、任务寄存器和 LDTR 指出了保护模式下存储器管理所使用的数据结构在内存中的位置。

(10) * 调试寄存器。DR_0～DR_7 用来控制和监视处理器的调试操作。

(11) * 机器检测寄存器。这些寄存器用于检测和报告硬件错误。

(12) * 存储器类型范围寄存器(MTRRs)。MTRRs 用于为物理存储器的范围指定存储器类型。

(13) * 机器相关寄存器(MSRs)。这些寄存器用来控制和报告处理器的性能,它们不能被应用程序所访问(除了时间戳计数器外)。

(14) * 性能监视寄存器。性能监视寄存器用于监视处理器性能事件。

(注:带 * 的资源在图中没有画出来。)

在 Pentium 4 的内部寄存器中,通用寄存器、段寄存器、指令指针寄存器与 80386 完全相同,标志寄存器也只是增加了几个状态位。图 2-28 是 Pentium 4 标志寄存器各标志位的情况。可以看出,Pentium 4 的标志位仅比 80386 增加了 4 位:AC、VIF、VIP 和 ID。

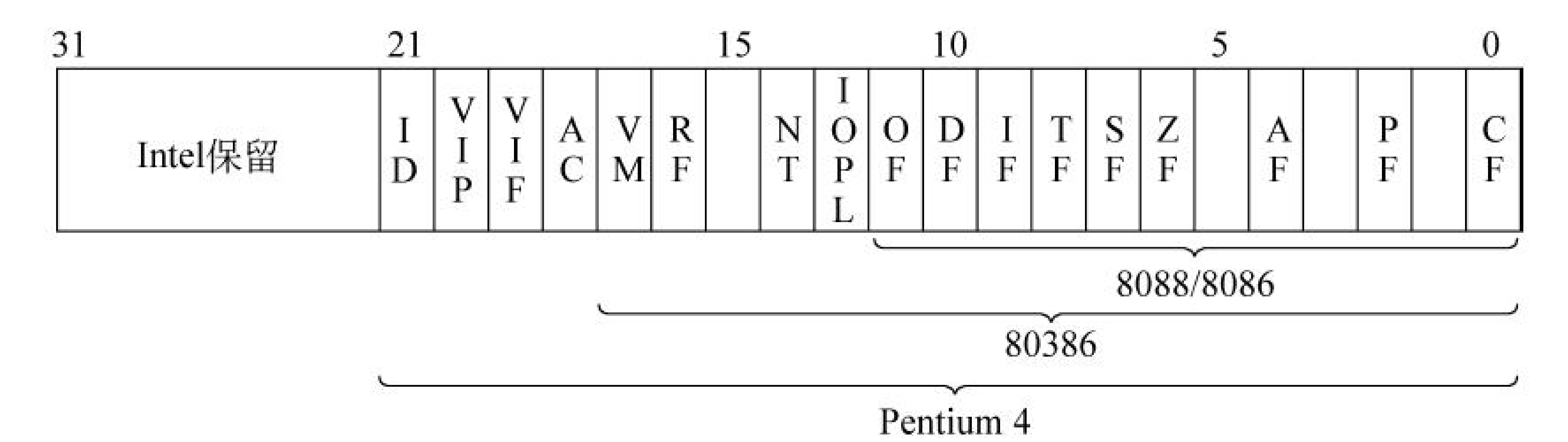


图 2-28 Pentium 4 的标志寄存器 EFALGS

(1) AC:对齐检查标志。当 AC=1 并且 CR_0 寄存器的 AM=1 时,允许存储器对齐检查。

(2) VIF:虚拟中断标志。VIF 标志位为 IF 标志位的虚拟映像,与 VIP 联合使用。

(3) VIP:未决虚拟中断标志。VIP=1 时表示有未决的中断,VIP=0 时表示没有未决的中断。

(4) ID:鉴别标志。如果程序能设置或清除这一位,表示可以使用 CPU ID 指令。

FPU 寄存器、MMX 寄存器和 XMM 寄存器可参考有关文献。

2.5 总 线

微型计算机从其诞生以来就采用了总线结构。CPU 通过总线实现读取指令并实现与内存、外设之间的数据交换,在 CPU、内存与外设确定的情况下,总线速度是制约计算

机整体性能的关键,总线的性能对于解决系统瓶颈、提高整个微机系统的性能有着十分重要的影响。因此在微型计算机 20 多年的发展过程中,总线的结构也在不断地发展和变化。

采用总线结构在系统设计、生产、使用和维护上有很多优越性,概括起来有以下几点。

(1) 便于采用模块结构设计方法,简化系统设计。

(2) 标准总线可以得到多个厂商的广泛支持,便于生产与之兼容的硬件板卡和软件。

(3) 模块结构方式便于系统的扩充和升级。

(4) 便于故障诊断和维修,同时也降低了成本。

2.5.1 概述

1. 总线的概念

总线是一组信号线的集合,是计算机系统各部件之间传输地址、数据和控制信息的公共通路。从物理结构来看,它由一组导线和相关的控制、驱动电路组成。在微型计算机系统中,总线常被作为一个独立部件看待。

总线的特点在于其公用性,即它可同时挂接多个部件或设备(对于只连接两个部件或设备的信息通道,不称为总线)。总线上任何一个部件发送的信息都可被连接到总线上的其他所有设备接收到,但某一个时刻只能有一个设备进行信息传送。所以,当总线上挂接的部件过多时,就容易引起总线争用,总线对信号响应的实时性降低。

总线一般由多条通信线路组成,每一路信号线能够传送一位二进制 0 或 1,8 条信号线就能在同一时间并行传送一个字节的信息。

2. 总线的分类

计算机系统中含有多种类型的总线,可以从不同的角度进行分类。

1) 按传送信息的类型划分

从传送信息的类型上,总线可分为数据总线(DB,Data Bus)、地址总线(AB,Address Bus)及控制总线(CB,Control Bus)。

(1) 数据总线。数据总线是计算机系统内各部件之间进行数据传送的路径。数据总线的传送方向是双向的,可以由处理器发向其他部件,也可由其他部件将信号送向处理器。

数据总线一般由 8 条、16 条、32 条或更多条数据线组成,这些数据线的条数称为数据总线的宽度。由于每一条数据线一次只能传送一位二进制码,因此数据线的条数(即数据总线的宽度)就决定了每一次能同时传送的二进制位数。如果数据总线宽度为 8 位,指令的长度为 16 位,则取一条指令需要访问两次存储器。由此可以看出,数据总线的宽度是表现系统整体性能的关键因素之一。8088 CPU 的外部数据总线宽度为 8 位,而 Pentium CPU 的数据总线宽度为 64 位,大大加快了对存储器的存取速度。

(2) 地址总线。地址总线用于传送地址信息,即这类总线上所传送的一组二进制 0

或1表示的是某一个内存单元地址或I/O端口地址。它规定了数据总线上的数据来自于何处或被送往何处。例如,当CPU要从存储器中读取一个数据时,不论该数据是8位、16位或32位,都需要先形成存放该数据的地址,并将地址放到地址总线上,然后才能从指定的存储器单元中取出数据。因地址信息均由系统产生,所以它的传送方向是单向的。

地址总线的宽度决定了能够产生的地址码的个数,从而也就决定了计算机系统能够管理的最大存储器容量。除此之外,在进行输入输出操作时,地址总线还要传送I/O端口的地址。由于寻址I/O端口的容量要远低于内存的容量,所以一般在寻址端口时,只使用地址总线的低端几位,寻址内存时才使用地址总线的所有位。例如,在8086系统中,寻址端口时需要用到地址总线的低16位,高4位设定为“0”;寻址内存时则用全部20位地址信号。

(3) 控制总线。控制总线用于传送各种控制信号,以实现对数据总线、地址总线的访问及使用情况进行控制。控制信号的作用是在系统内各部件之间发送操作命令和定时信号,通常包括以下几种类型。

① 写存储器命令。在写存储器命令的控制下,数据总线上的数据被写入指定的存储器单元。

② 读存储器命令。在读存储器命令的控制下,将指定存储器单元中的数据放到数据总线上。

③ I/O写命令。在I/O写命令的控制下,将数据总线上的数据写入指定的I/O端口。

④ I/O读命令。在I/O读命令控制下,将指定I/O端口的数据放上数据总线。

⑤ 传送响应。传送响应用于表示数据已经被接收或已经将数据放上数据总线的应答信号。

⑥ 总线请求。总线请求用于表示系统内的某一部件欲获得对总线的控制权的信号。

⑦ 总线响应。总线响应表示获准系统内某部件控制总线。

⑧ 中断请求。中断请求表示系统内某中断源发出欲中断的请求信号。

⑨ 中断响应。中断响应表示系统内某中断源发出的中断请求信号已获得响应。

⑩ 时钟和复位。时钟信号用于同步操作时的同步控制。在初始化操作时,需要用复位命令。

控制信号从总体上讲,其传送方向是双向的,但就某一具体信号来讲,其信息的走向都是单向的。

2) 按总线的层次结构划分

总线按照层次结构可分为前端总线(或CPU总线)、系统总线和外设总线。计算机系统内各层的信息传送由各层总线完成。

(1) 前端总线。前端总线包括地址总线、数据总线和控制总线,一般是指从CPU引脚上引出的连接线,用来实现CPU与主存储器、CPU与I/O接口芯片、CPU与控制芯片组等芯片之间的信息传输,也用于系统中多个CPU之间的连接。前端总线是生产厂家针对其具体的处理器设计的,与具体的处理器有直接的关系,没有统一的标准。

(2) 系统总线。系统总线也称为I/O通道总线,同样包括地址总线、数据总线和控制

总线，是主机系统与外围设备之间的通信通道。在主板上，系统总线表现为与 I/O 扩展插槽引线连接的一组逻辑电路和导线。I/O 插槽上可插入各种扩展板卡，它们作为各种外部设备的适配器与外设相连。系统总线有统一的标准，各种外设适配卡可以按照这些标准进行设计。所以，各种总线标准主要是指系统总线的标准以及与系统总线相连的插槽的标准。常见的系统总线标准有：ISA(Industry Standard Architecture)总线、PCI(Peripheral Component Interconnect)总线、AGP(Accelerated Graphics Port)总线等。

(3) 外设总线。外设总线是指计算机主机与外部设备接口的总线，实际上是一种外设的接口标准。目前在微型计算机上流行的接口标准有 IDE(EIDE)、SCSI、USB 和 IEEE 1394 这 4 种。前两种主要是与硬盘、光驱等 IDE 设备接口，后两种新型外部总线可以用来连接多种外部设备。

除以上两种分类原则外，总线还可以按其相对于 CPU 的位置分为片内总线和片外总线。在 CPU 内部，寄存器、算术逻辑部件 ALU、控制部件以及地址形成部件之间进行信息传送所用的总线称为片内总线(即芯片内部的总线)；而通常所说的总线则是指片外总线，是 CPU 与内存和输入输出设备接口之间进行通信的通路。有的资料上也把片内总线叫做内部总线或内总线(Internal Bus)，把片外总线叫做外部总线或外总线(External Bus)。

3. 总线结构

在微机系统中，总线结构可划分为两种，即单总线结构和多总线结构。在多总线结构中，又以双总线结构为主。

1) 单总线结构

本书第 1 章中图 1-5 所示的微机系统结构就属于单总线结构。计算机的各个部件均挂接在一组总线上，构成微机的硬件系统，所以它又称为面向系统的单总线结构。在单总线结构中，CPU 与主存之间、CPU 与 I/O 设备之间、I/O 设备与主存之间、各种设备之间都通过单一系统总线交换信息。

单总线结构的优点是控制简单、扩充方便。但由于所有设备部件均挂接在单一总线上，使这种结构只能分时工作，即同一时刻只能在两个设备之间传送数据，这就使系统总体数据传输的效率和速度受到限制，这是单总线结构的主要缺点。

2) 多总线结构

(1) 双总线结构。双总线结构又分为面向 CPU 的双总线结构和面向存储器的双总线结构。

面向 CPU 的双总线结构如图 2-29 所示。其中一组总线是 CPU 与主存储器之间进行信息交换的公共通路，称为存储总线。另一组是 CPU 与 I/O 设备之间进行信息交换的公共通路，称为输入输出(I/O)总线。外部设备通过挂接在 I/O 总线上的接口电路与 CPU 交换信息。

由于在 CPU 与主存储器之间、CPU 与 I/O 设备之间分别设置了总线，从而提高了微机系统信息传送的速率。但是由于外设与主存之间没有直接的通路，它们之间的信息交换必须通过 CPU 才能进行中转，这就要求 CPU 必须花大量的时间来进行信息的输入输

出处理，从而降低了 CPU 的工作效率(或增加了 CPU 的占用率)。一般来说，外设工作时要求 CPU 干预得越少越好。CPU 干预得越少，这个设备的 CPU 占用率就越低，说明设备的智能化程度越高。CPU 占用率与系统结构有很大关系，这是面向 CPU 的双总线结构的主要缺点。

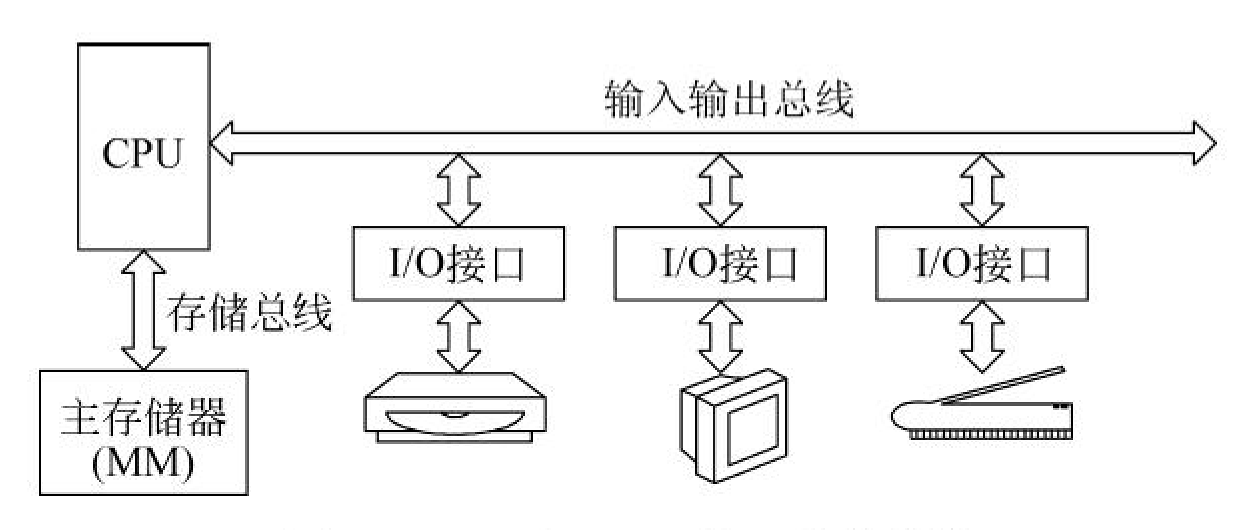


图 2-29 面向 CPU 的双总线结构

面向存储器的双总线结构保留了单总线结构的优点，即所有设备和部件均可通过总线交换信息。与单总线结构不同的是，在 CPU 与主存储器之间又专门设置了一条高速总线，使 CPU 可以通过它直接与主存储器交换信息。面向主存储器的双总线结构不仅使信息传送效率提高，而且减轻了总线的负担，这是它的主要优点。但这种总线结构硬件造价较高，高档微机中通常采用这种面向存储器的双总线结构。图 2-30 是这种结构的示意图。

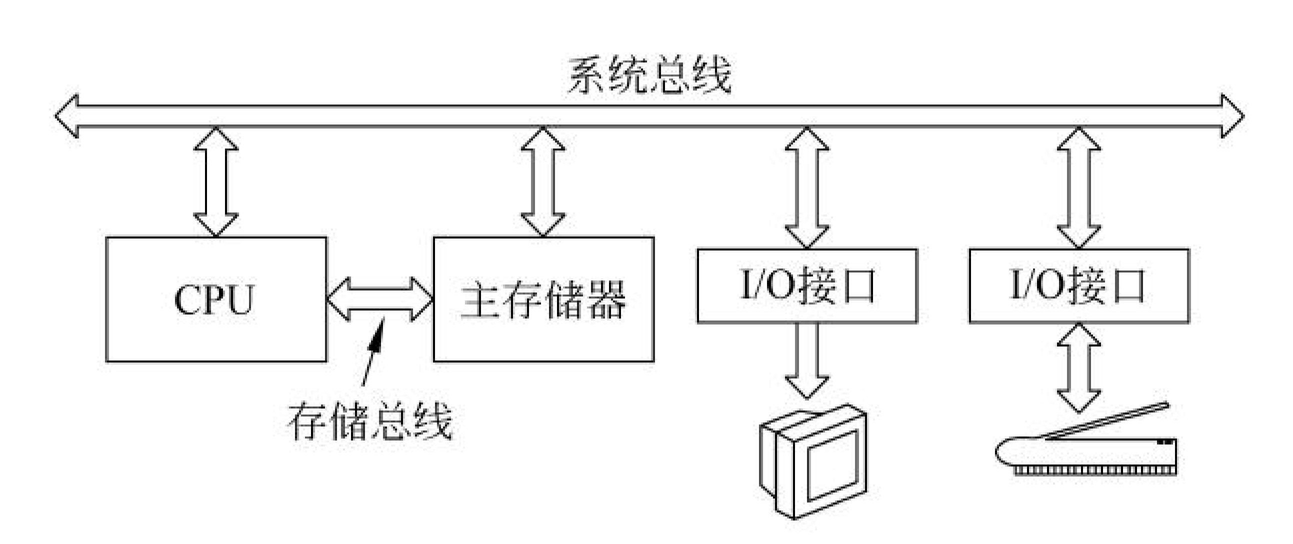


图 2-30 面向主存储器的双总线结构

(2) 多总线结构。随着对微机性能的要求越来越高，现代微机的体系结构已不再采用单总线或双总线的结构，而是采用更复杂的多总线结构，如图 2-31 所示。

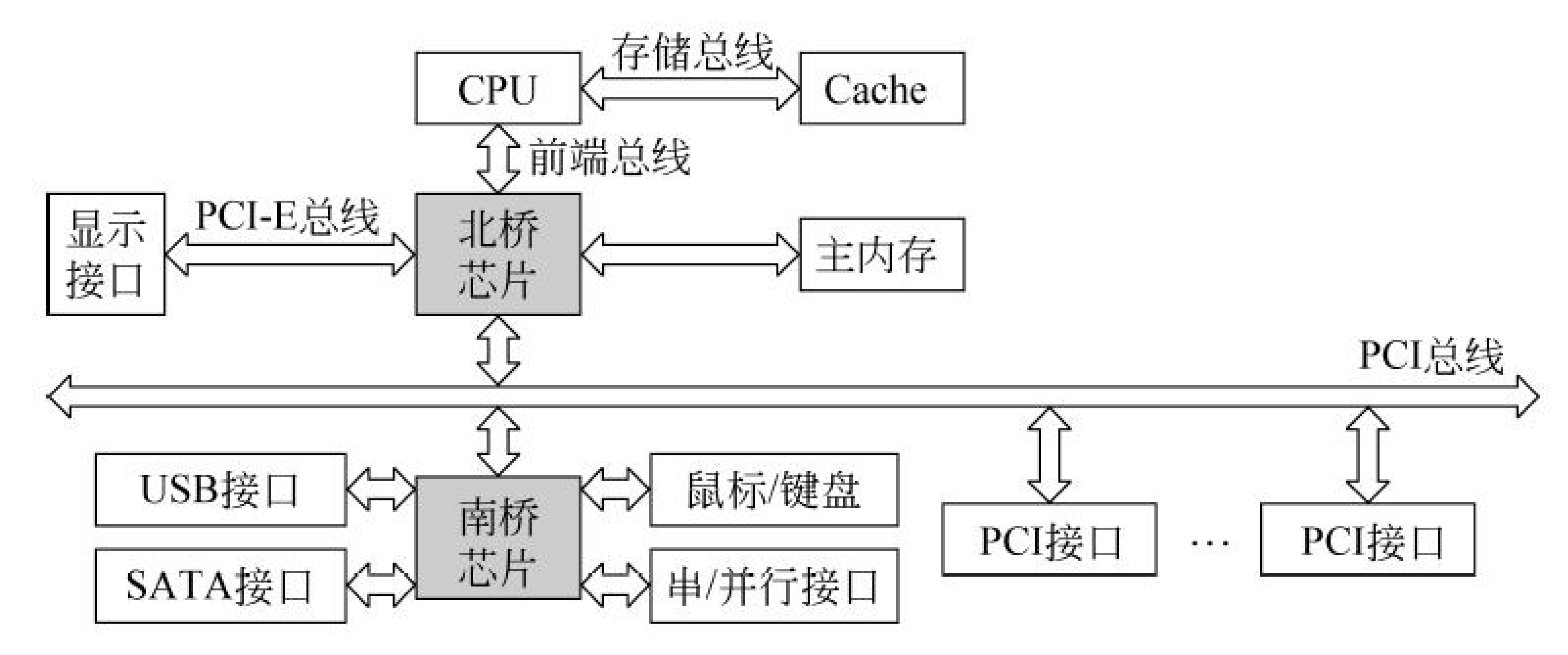


图 2-31 现代微机中的多总线结构

4. 总线操作

接到总线上的设备有两种工作方式：主控方式和从属方式。由此，连接到总线上的设备就分为主控设备和从属设备。主控设备可以通过总线进行数据传送；从属设备只能按主控设备的要求工作，接收传送过来的数据。

微机系统中的各种操作，包括处理器内部寄存器操作、处理器对存储器的读写操作、处理器对 I/O 端口的读写操作、中断操作、直接存储器存取操作等，都是通过总线进行信息交换的，它们在本质上都是总线操作。总线操作的特点是：任意时刻，总线上只能允许一对设备(主控设备和从属设备)进行信息交换。当有多个设备要使用总线时，只能分时使用，即将总线时间分为若干段，每一个时间段完成设备间的一次信息交换，包括从主控设备申请使用总线到数据传送完毕。这个时间段称为一个数据传送周期或总线操作周期。

一个总线周期分为 5 个步骤：总线请求、总线仲裁、寻址、传送数据和传送结束。

(1) 总线请求。总线请求是由使用总线的主控设备向总线仲裁机构提出使用总线的请求。

(2) 总线仲裁。总线仲裁决定在下一个传送周期由哪个请求源使用总线。

(3) 寻址。寻址是指取得总线使用权的主控设备，通过地址总线发出本次要传送的数据的地址及相关命令，通过译码使参与本次数据传送的从属设备被选中。

(4) 数据传送。数据传送实现从主控设备到从属设备的数据传送。

(5) 传送结束。传送结束是指主控设备、从属设备的相关信息均从总线上撤除，让出总线，以使其他设备能继续使用总线。

对于只有一个主控设备的单处理器系统，不存在总线请求、仲裁和撤除问题，总线始终归它所有，此时的总线周期只有寻址和传送两个阶段。在包括中断控制器、DMA 控制器及多处理器系统中，则需要专门的仲裁机构来分配总线的控制和使用权。

5. 总线的主要性能指标

1) 总线的带宽

总线的带宽指的是单位时间内总线上可传送的数据量，即人们常说的每秒钟传送多少字节，单位是字节/秒(B/s)或兆字节/秒(MB/s)。与总线带宽密切相关的两个概念是总线的宽度和总线的工作频率。

2) 总线的位宽

总线的位宽指的是总线能同时传送的数据位数，即人们常说的 16 位、32 位、64 位等总线宽度的概念。在工作频率固定的条件下，总线的带宽与位宽成正比。

3) 总线的工作频率

总线的工作频率也称为总线的时钟频率，以 MHz 为单位。它是指用于协调总线上的各种操作的时钟信号的频率。工作频率越高则总线工作速度越快，也即总线带宽越宽。

总线带宽、总线宽度、总线工作频率三者之间的关系就像高速公路上的车流量、车道数和车速的关系。车流量取决于车道数和车速，车道数越多、车速越快，则车流量越大。

同样，总线带宽取决于总线宽度和工作频率，总线宽度越宽，工作频率越高，则总线带宽越大。当然，单方面提高总线的宽度或工作频率都只能部分提高总线的带宽，并容易达到各自的极限。只有两者配合才能使总线的带宽得到更大的提升。

总线带宽的计算公式如下：

总线带宽 BW=(总线宽度/8)×总线时钟频率/每个存取周期的时钟数

例如，总线时钟频率为 66MHz 的 32 位总线，若每两个时钟周期完成一次总线存取操作，则总线带宽=32/8×66/2=132MB/s。

2.5.2 总线的基本功能

总线传输需要解决以下几方面的问题。

(1) 总线传输同步。为使信息正确传送，防止丢失，需对总线通信进行定时，根据定时方式不同，可分为同步、异步及半同步 3 种数据传送方式。

(2) 总线仲裁控制。在总线上某一时刻只能有一个总线主控部件控制总线，为避免多个部件同时发送信息到总线的矛盾，需要有总线仲裁机构。

(3) 出错处理。数据传送过程中可能产生错误，有些接收部件有自动纠错能力，可以自动纠正错误；有些部件虽无自动纠错能力，但能发现错误，这时可发出“数据出错”信号，通知 CPU 来进行处理。

(4) 总线驱动。在计算机系统中通常采用三态输出电路或集电极开路输出电路来驱动总线。后者速度较低，常用在 I/O 总线上。

因此，总线的基本功能包括数据传送、仲裁控制、出错处理及总线驱动。

1. 总线的数据传送

数据在总线上传送时，为确保传送的可靠性，传送过程必须由定时信号控制，定时信号使主控设备和从属设备之间的操作同步。定时实现的方式有 3 种：同步定时方式、异步定时方式和半同步定时方式。

1) 同步定时方式

采用同步定时方式时，总线上的数据传送用一个公共的时钟来同步双方的操作，发送和接收信号都在固定的时刻发出。图 2-32 是总线执行写操作时的定时图。主控设备(源端)于某一时刻在数据准备好信号 READY 的控制下将数据发出，从属设备(目的端)在接收信号 ACK 控制下接收数据。同步定时方式比异步定时方式的吞吐量大，因为在源端和目的端之间不需要有来往传送的“握手”控制信号，但延迟时间 t_1 和 t_2 要根据接到总线上最慢的设备来设定。

同步总线定时方法的缺点是源部件无法知道目的部件是否已收到数据，目的部件也无从知道源部件的数据是否已真正送到总线上。8088 系统中的总线若不考虑插入的等待周期，基本上都属于同步定时方式。

2) 异步定时方式

异步定时方法中没有固定的时钟，定时序列中的每一步都要靠信号在源端和目的端

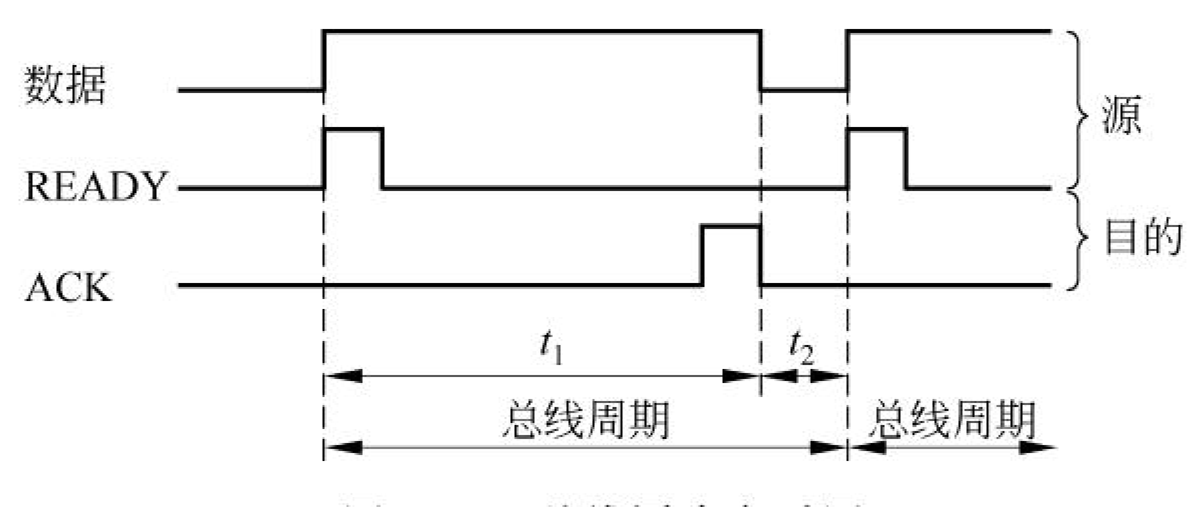


图 2-32　总线同步定时图

间的来回传送来实现。这些控制信号的传送要有相当可观的延迟时间。为减少信号传送的复杂性,在异步方式中并不是每一步都靠信号传递来定时的,而是把某几步改用等待一段足够长的固定延迟时间来代替对方传送过来的信号。这种用固定延迟时间的信号叫做隐含信号,根据隐含信号的多少,可以把异步总线定时分为非互锁的、半互锁的和全互锁的 3 种方式。这里仅介绍一下非互锁异步定时方式,如图 2-33 所示。在这个方式中,READY 信号和 ACK 信号的脉冲宽度设定为固定时间,即 t_2 和 t_4 为定值。数据送到总线上,经过延迟时间 t_1 后,源端把 READY 信号升高,目的端收到 READY 信号后,接收总线上的数据,并经 t_3 时间后使 ACK 升高以此通知源端,源端接收到 ACK 后,经 t_5 时间从总线上撤去数据,再用 t_6 时间使总线状态稳定,然后开始下一个总线周期。

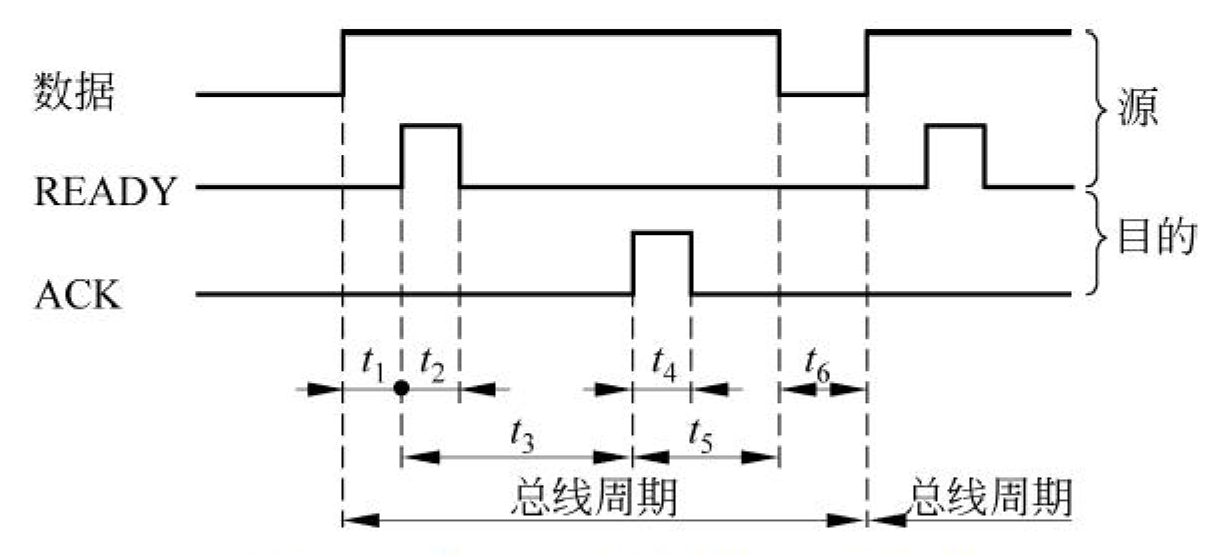


图 2-33　非互锁异同步总线的定时图

这种方式的优点是任何速度的设备之间都能互相进行通信。缺点是延迟较大。

3）半同步定时方式

半同步总线定时方式仍利用时钟脉冲的边沿判断某一信号的状态,或控制某一信号的产生和消失,使传输操作与时钟同步。每个动作只能在固定时钟确定的一定时刻发生。它不像同步传输那样传输周期固定,其控制信号间的间隔时间根据总线上所挂接设备的快慢程度是可变的,但间隔时间必须是时钟周期的整倍数。控制周期间隔时间的方法是慢速从属设备通过一根状态信号线(如 WAIT、READY)通知主控设备增加若干时钟周期。WAIT 线有效(或 READY 线无效)时,表示从属设备未准备好接收数据或未把数据送到数据线上。CPU 若检测到这个状态,便自动地在总线周期中插入一个时钟周期,等待从属设备准备好。从属设备准备好后撤销该状态信号,CPU 才不再延长当前的传输周期。

半同步方式允许不同速度的部件协同工作,主控设备可以根据从属设备的状态来自动延长总线时钟周期,但改变后的总线周期一定是时钟周期的整数倍,这是与异步方式的不同之处(异步方式的总线周期长度是完全任意的)。

8086 CPU 的总线周期插入等待状态就是半同步方式的一个实例。它于 T_3 前沿检测 READY 状态，若从属设备未准备好，则在 T_4 之前插入一个或多个等待状态 T_W，直至从属设备准备好才结束等待状态，进入 T_4 周期。

2. 总线仲裁控制

总线仲裁也叫总线判优。由于总线为多个部件所共享，在总线上某一时刻只能有一个总线主控部件控制总线，为了正确地实现多个部件之间的通信，避免各部件同时发送信息到总线的冲突，必须要有一个总线仲裁机构，对总线的使用进行合理的分配和管理。

当总线上的一个部件要与另一个部件进行通信时，首先应该发出请求信号。在某一时刻，可能有多个部件同时要求使用总线，总线仲裁控制机构根据一定的判决原则，决定首先由哪个部件使用总线。只有获得了总线使用权的部件才能开始传送数据。

根据总线控制部件的位置，控制方式可以分成集中方式与分散方式两类。总线控制逻辑集中在一处的，称为集中式总线控制；总线控制逻辑分散在总线各部件中的，称为分散式总线控制。以下简单介绍集中式控制方式，分散式控制方式参考有关资料。

集中式控制方式主要有以下 3 种。

1）链式查询方式

链式查询方式如图 2-34 所示。图中所示的总线控制部件在单总线系统和三总线系统中常常是 CPU 的一部分；在双总线系统的 I/O 总线中，它是通道的一部分。链式查询方式需要有 3 根控制线。

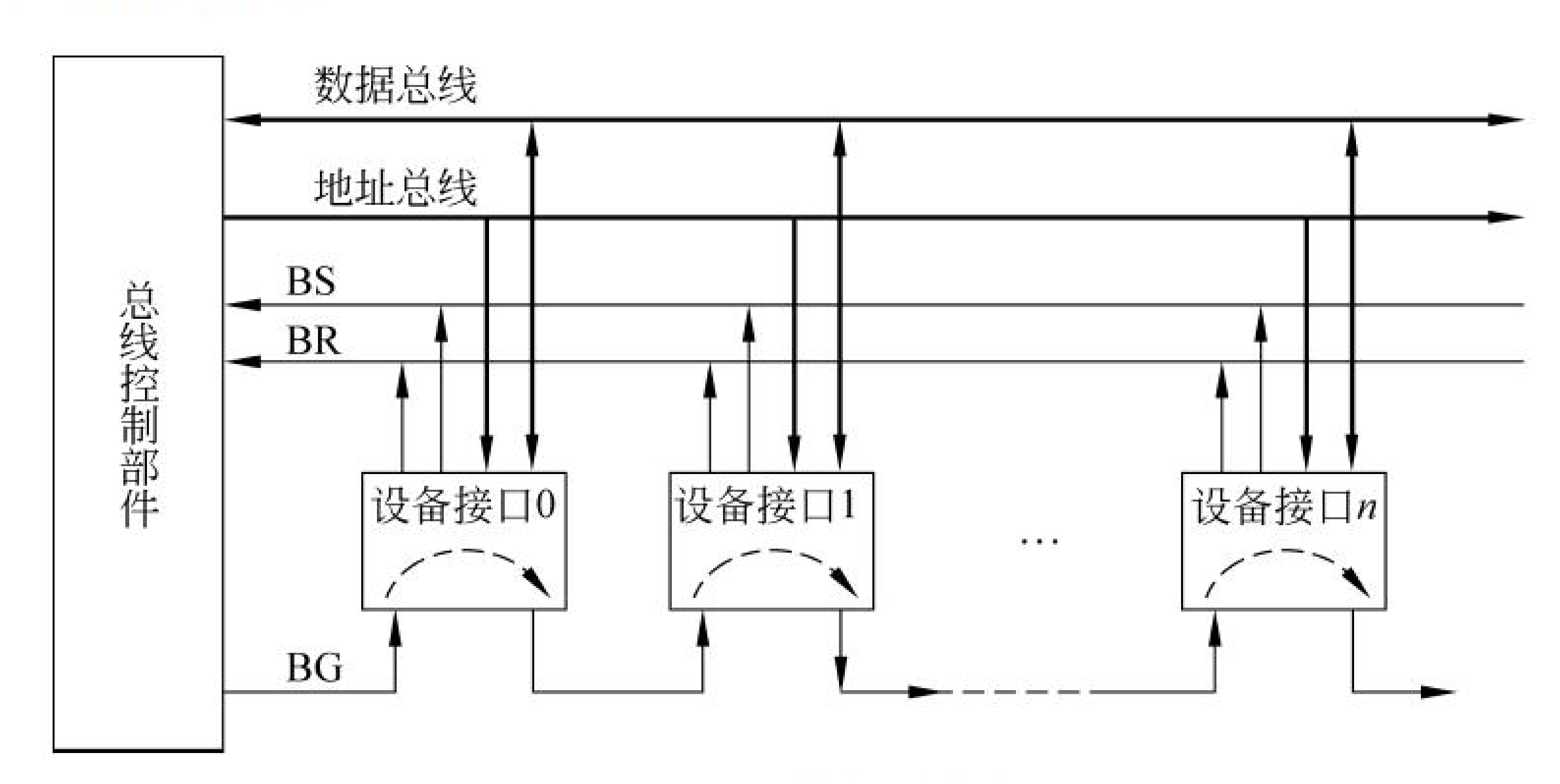


图 2-34　链式查询方式

（1）总线忙信号 BS：该信号有效时，表示总线正被某外设使用。

（2）总线请求信号 BR：该信号有效时，表示至少有一个外设请求使用总线。

（3）总线回答信号 BG：该信号有效时，表示总线控制部件响应了外设的总线请求。

链式查询方式的主要特征是：总线回答信号 BG 的传送是串行地址从一个 I/O 接口送到下一个 I/O 接口。假如 BG 到达的接口无总线请求，则继续往下传；假如 BG 到达的接口有总线请求，BG 信号便不再往下传，这意味着该 I/O 接口就获得了总线使用权。

显然，在查询链中离总线控制器最近的设备具有最高优先权，离总线控制器越远优先权越低。因此，链式查询是通过接口的优先权排队电路来实现的。

链式查询方式的优点是：只用很少几根线就能按一定的优先次序实现总线控制，并且这种链式结构很容易扩充设备。链式查询方式的缺点是对询问链的电路故障很敏感，如果第 i 个设备的接口中有关链的电路有故障，那么第 i 个以后的设备都不能进行工作。另外，查询链的优先级是固定的，如果优先级高的设备出现频繁的请求时，那么优先级较低的设备可能长时间请求不到总线。

2）计数器查询方式

计数器查询方式原理如图 2-35 所示。总线上任何设备要求使用总线时，都通过 BR 线发出总线请求。总线控制器接到总线请求信号后，在 BS 线为 0 的情况下让计数器开始计数，计数值通过一组设备地址线发向各设备。每个外设接口都有一个设备地址判别电路，当设备地址线上的计数值与请求使用总线的设备地址一致时，该设备就获得了总线使用，并置 BS 线为 1，此时中止计数查询。

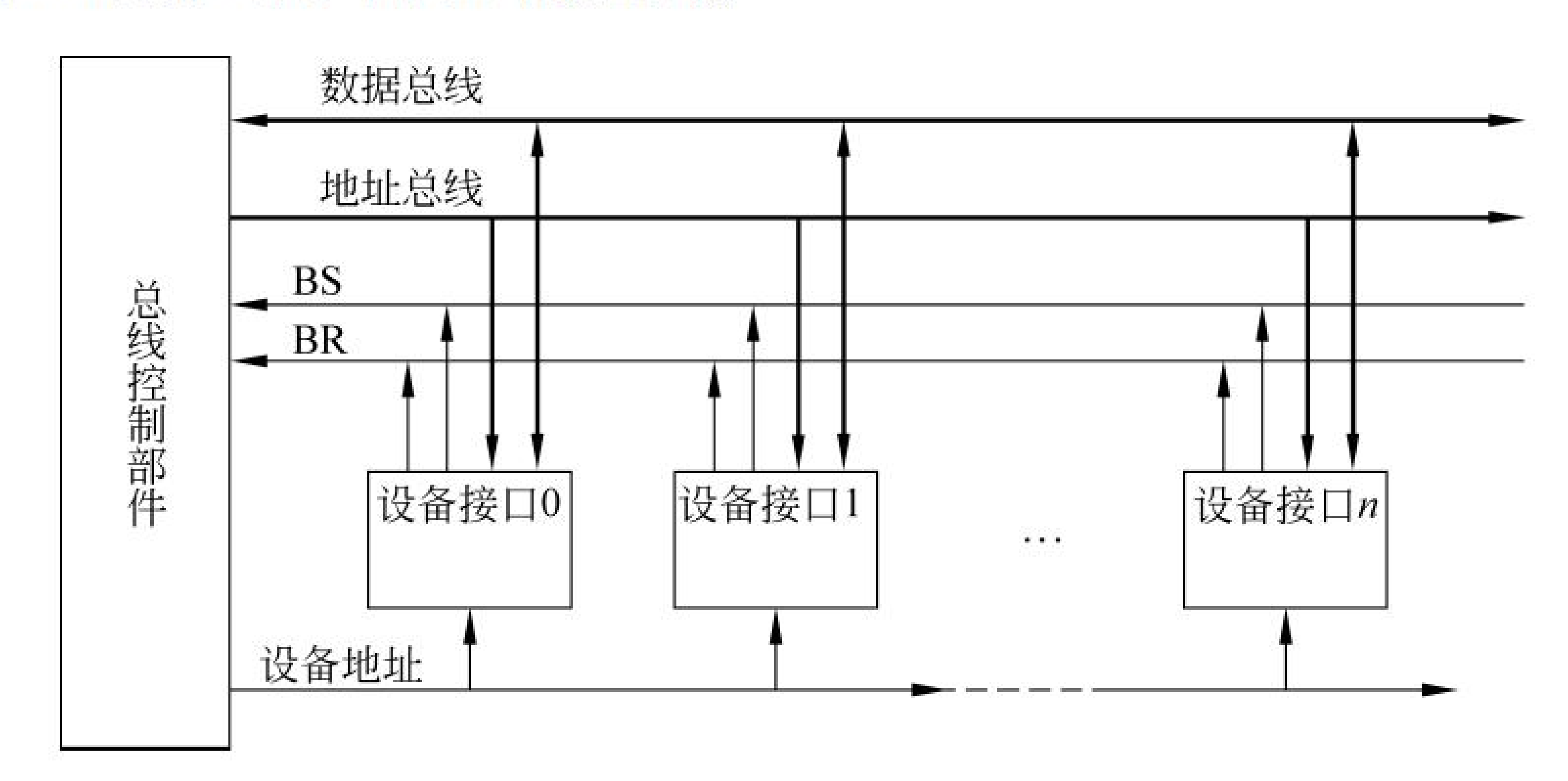


图 2-35　计数器查询方式

每次计数可以从 0 开始，也可从上次计数的中止点开始。如果从 0 开始，各设备的优先次序与链式查询相同，优先级的顺序是固定的；如果从中止点开始，则每个设备使用总线的优先级别是相等的。计数器的初值也可以用程序来设置，这就可以方便地改变优先次序，显然这种灵活性是以增加相应的控制线数为代价的。

3）独立请求方式

独立请求方式原理如图 2-36 所示。在独立请求方式中，每一个设备均有独立的总线请求线 BR_i 和总线回答线 BG_i。当设备要求使用总线时，便发出该设备的请求信号 BR_i。总线控制部件中一般有一个排队电路，根据一定的优先次序决定首先响应哪个设备的请求，若响应了该设备的请求则发出总线回答信号 BG_i。

独立请求方式的优点是响应时间快，即为确定优先响应设备所花费的时间少，不用逐个查询设备，然而这是以增加控制线数为代价的。在链式查询中仅用两根线确定总线使用权属于哪个设备；在计数查询中大致用 $\log_2 n$ 根线（其中 n 是允许接纳的最大设备数）。而独立请求方式需则采用 $2n$ 根线。

独立请求方式对优先次序的控制也是相当灵活的。它可以预先固定，如让 BR_0 优先级最高、BR_1 次之……BR_n 最低；或者通过程序来改变优先次序；或者采用屏蔽某个请求的办法，不响应来自与当前处理无关的设备的请求。

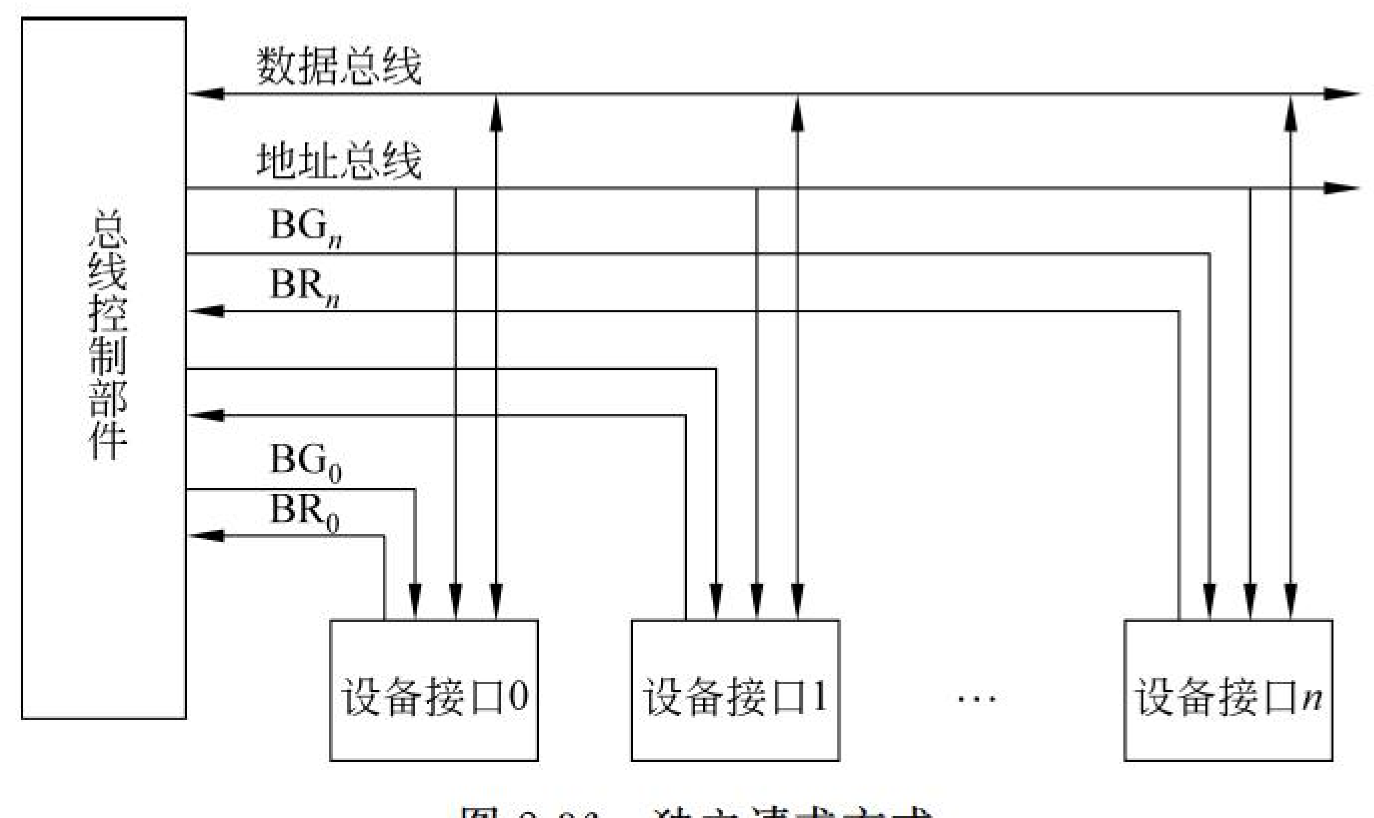


图 2-36　独立请求方式

3. 总线驱动及出错处理

1）总线驱动

在计算机系统中，总线上连接的设备接口很多，每个接口电路都要从总线上吸收电流，因此需要总线驱动。常用的总线驱动器是三态总线驱动器。但总线驱动器的驱动能力有限，因此在扩充外设接口时要加以注意，通常一个模块或部件限制为 1～2 个负载（必须是低功耗的负载），同时为减轻总线上的负载，在设备接口电路与总线之间通常要设置缓冲器，起隔离和驱动的作用。如果所有接口都直接连接到总线，将会因总线上负载过于沉重而使系统无法正常工作。

2）出错处理

数据传送过程中可能产生错误，解决的方法是在传输的数据中增加一些冗余位，使冗余位与传送的数据具有某种特殊的关系。例如，使数据中 1 的个数为偶数，这样接收部件中的错误校验电路就可以检查出接收的数据是否出错。若这种特殊关系存在，表示接收的数据正确；若这种特殊关系不存在，则表示接收的数据出错。发现错误后，如何去处理错误？通常有两种方法，当总线控制器和设备接口中的总线接口部件有自动纠错电路时，纠错电路可以根据错误的状态用某种算法自动纠正错误；若部件中无自动纠错电路，则可在发现错误后发出“数据出错”信号让 CPU 来进行错误处理，通常是向 CPU 发出中断请求信号，CPU 响应中断后，转入错误处理程序来处理异常情况。

2.5.3　常用系统总线和外设总线标准

1. 系统总线

1）系统总线标准

在国际化生产非常流行的今天，一台计算机往往不再是由单一的企业按大而全的方式生产出来，而是将计算机中的各部件交给不同的专业化生产厂家分别生产，然后再由组装厂组装成完整的计算机。这样做主要是为了降低成本，提高生产率和产品的质量。为

了将不同厂家生产的各种部件组装在一起，形成一台完整的计算机，需要各厂家按照一定的标准进行生产，特别是系统总线。由于外设接口卡都要通过它接入系统，所以总线标准的制订更显重要。系统总线制订的标准有很多，例如ISA、EISA、MCA、PCI-E、PCI、AGP等。

(1) ISA(Industry Standard Architecture)工业标准总线是IBM公司为286/AT微型计算机制定的一种总线标准，也称为AT总线标准。随着技术的发展，作为8/16位的总线标准，ISA总线已基本被淘汰。

(2) MCA(Micro Channel Architecture)微通道总线结构是IBM公司专为其PS/2系统开发的总线标准。由于执行的是使用许可证制度，因此未能得到有效推广。

(3) EISA(Extended Industry Standard Architecture)是在ISA总线基础上为32位CPU设计的扩展工业标准总线。

(4) PCI(Peripheral Component Interconnect)是SIG(Special Interest Group)集团推出的高性能的总线结构。1992年起，先后有Intel、HP、IBM、Apple、DEC、Compaq、NEC等著名的厂商加盟重新组建。

(5) AGP(Accelerated Graphics Port)加速图形接口总线是一种专为提高视频带宽而设计的总线规范。

(6) PCI-E(PCI Express)总线是目前最新的系统总线标准。虽然是在PCI总线的基础上发展起来的，但它与并行体系的PCI没有任何相似之处。它采用串行方式传输数据，依靠高频率来获得高性能，因此PCI Express也一度被人们称为“串行PCI”。

系统总线与I/O接口卡的连接是用总线插座来实现的，即各I/O接口插件板连入系统时需要插入与系统总线连接的插槽。为使不同厂家生产的I/O接口板都可以连入系统后正常工作，就需要制定相应的总线标准。

系统总线通常为50～100根信号线，这些信号线可分为5个主要类型。

(1) 数据线：决定数据宽度。

(2) 地址线：决定直接选址范围。

(3) 控制线：包括控制、时序和中断线，决定总线功能和适应性的好坏。

(4) 电源线和地线：决定电源的种类及地线的分布和用法。

(5) 备用线：留给厂家或用户自己定义。

有关这些信号线的标准主要涉及如下几个方面：信号的名称，信号的定时关系，信号的电平，连接插件的几何尺寸，连接插件的电气参数，引脚的定义、名称、序号，引脚的个数，引脚的位置，电源及地线等。

微型机自问世以来，从8位机到16位机、32位机一直发展到了64位机，为了适应数据宽度的增加和系统性能的提高，依次推出并采用的系统总线标准有XT总线、ISA总线、EISA总线、PCI总线以及专为提高视频带宽而设计的AGP总线。下面简单介绍一下8086 CPU以来使用最广泛的4种系统总线：ISA总线、PCI总线、AGP总线和PCI Express总线。

2) ISA总线

ISA(Industry Standard Architecture)是工业标准体系结构总线的简称，是由美国

IBM 公司推出的 16 位标准总线，数据传输率为 16Mb/s，主要用于 IBM-PC/XT、AT 及其兼容机上。

(1) ISA 总线的起源。最早的 PC 总线是 IBM 公司于 1981 年推出的基于 8 位机 PC/XT 的总线，称为 PC 总线。1984 年 IBM 公司推出了 16 位微型计算机 PC/AT，其总线称为 AT 总线。然而 IBM 公司从未将 AT 总线规格公布于众，这就给兼容设备生产商开发外设接口卡造成了很大的困难。为解决这个问题，Intel 公司、IEEE 和 EISA 集团联合开发了与 IBM/AT 原装机总线意义相近的 ISA 总线，即 8/16 位的工业标准体系结构(ISA，Industry Standard Architecture)总线。

(2) ISA 总线的主要特点和性能指标。8 位 ISA 扩展总线插槽由 62 个引脚组成，用于 8 位的插卡。8/16 位的扩展插槽除了具有一个 8 位 62 线的连接器外，还有一个附加的 36 线连接器，这种扩展总线插槽既可支持 8 位的插卡，也可支持 16 位插卡。ISA 总线的主要性能指标如下。

① I/O 地址空间为 0100H～03FFH。

② 24 位地址线可直接寻址的内存容量为 16MB。

③ 总线宽度 8/16 位，最高时钟频率为 8MHz，最大稳态传输率为 16Mb/s。

④ 支持 15 级中断。

⑤ 7 个 DMA 通道。

⑥ 开放式总线结构，允许多个 CPU 共享系统资源。

3) PCI 总线

PCI(外设互连，Peripheral Component Interconnect)总线是 1991 年由 Intel 公司提出，并联合其他多家公司共同推出的 32/64 位标准总线，是一种与 CPU 隔离的总线结构，能与 CPU 同时工作。这种总线适应性强、速度快，数据传输率为 133Mb/s，适用于 Pentium 以上的微型计算机。

(1) PCI 总线的主要性能和特点。PCI 总线是一种不依附于某个具体处理器的局部总线。从结构上看，PCI 是在 CPU 和原来的系统总线之间插入的另一级总线，具体由一个桥接电路(习惯上称为北桥芯片)实现对这一层的管理，并实现上下之间的接口以协调数据的传送。管理器提供了信号缓冲，使之能支持 10 种外设，并能在高时钟频率下保持高性能。PCI 总线也支持总线主控技术，允许智能设备在需要时取得总线控制权，以加速数据传送。PCI 总线的主要性能如下。

① 总线宽度为 32b/64b，总线时钟频率为 33MHz/66MHz，最大数据传输速率为 528Mb/s。

② 时钟同步方式。

③ 与 CPU 及时钟频率无关。

④ 能自动识别外设(即插即用功能)。

⑤ 具有与处理器和存储器子系统完全并行操作的能力。

⑥ 具有隐含的中央仲裁系统。

⑦ 采用多路复用(地址线和数据线)，减少了引脚数。

⑧ 完全的多总线主控能力。

⑨ 提供地址和数据的奇偶校验。

(2) PCI总线体系结构。PCI总线的体系结构如图2-37所示。

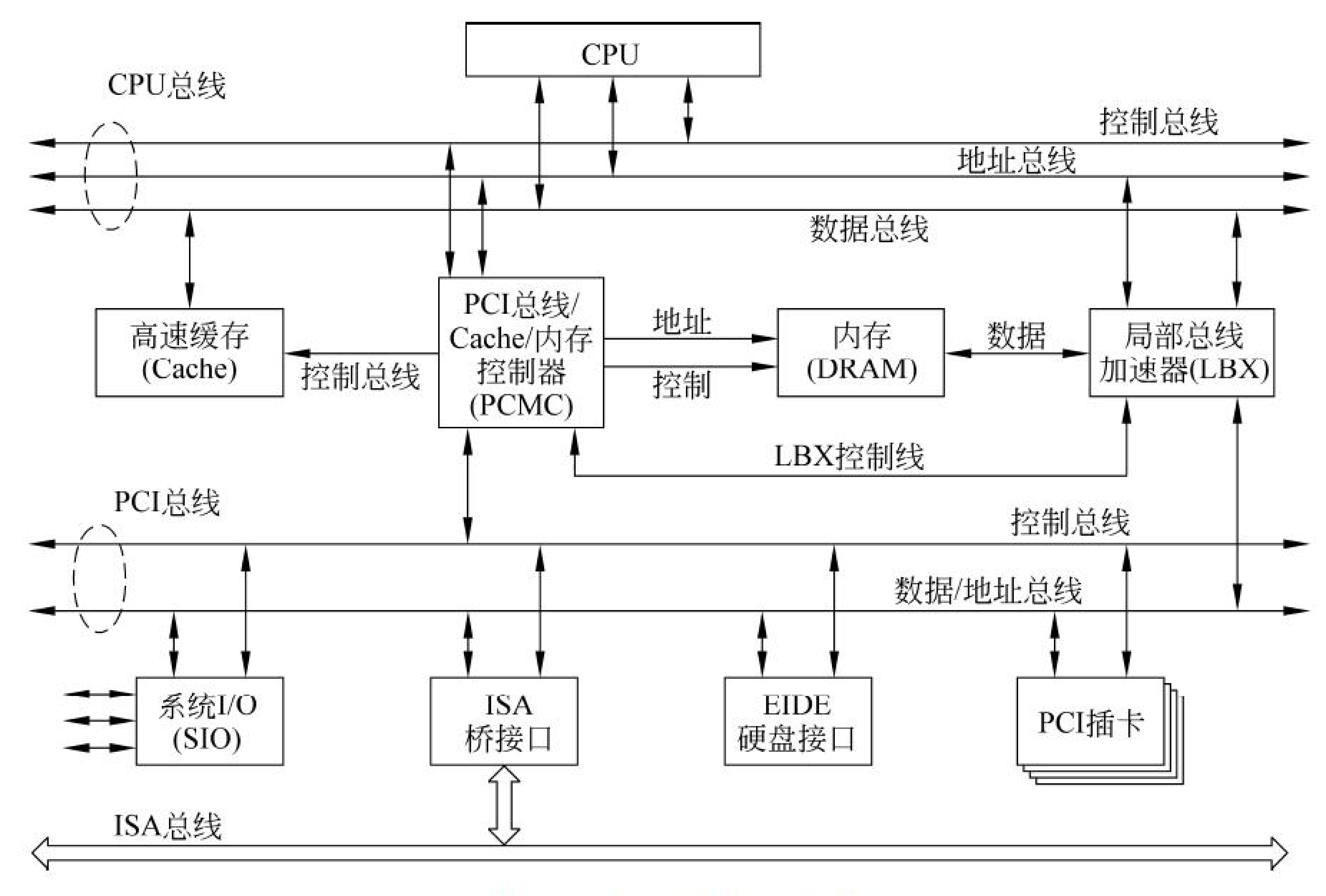


图2-37 PCI总线体系结构

从图中可以看到,CPU总线和PCI总线由桥接电路(PCMC)相连。芯片中除了含有桥接电路外,还有Cache控制器和DRAM控制器等其他控制电路。PCI总线上可挂接高速设备,如图形控制器、IDE设备或SCSI设备、网络控制器等。PCI总线和ISA/EISA总线之间也通过桥接电路(习惯上称为南桥芯片)相连,ISA/EISA上挂接传统的慢速设备,继承原有的资源。PCI总线把ISA/EISA总线作为一种外部设备与之进行数据交换。

此外,PCI总线还支持其他一些连接方式,如双PCI总线方式、PCI to PCI方式、多处理器服务器方式等。

4) AGP总线

(1) 设计AGP总线的目的。AGP(加速图形接口,Accelerated Graphics Port)总线是一种专为提高视频带宽而设计的总线规范。AGP插槽可以插入符合该规范的AGP显卡。其视频信号的传输速率可以从PCI的133Mb/s提高到266Mb/s(×1模式)、533Mb/s(×2模式)、1066Mb/s(×4模式)或2133Mb/s(×8模式)。严格说来,AGP不能称为总线,因为它仅在AGP控制芯片和AGP显卡之间提供了点到点的连接。

在AGP出现以前,几乎所有图形显卡都采用PCI总线接口。随着图形显卡3D图形处理性能的大幅度提升,显卡处理的数据越来越多,PCI接口就逐渐地暴露出它的局限性。这种局限性主要表现在3D图形描绘中,储存在PCI显卡显示内存中的不仅有影像数据,还有纹理数据(Texture Data)、Z轴的距离数据及Alpha变换数据等,特别是纹理数据的信息量相当大。例如,显示1024×768×16位真彩色的3D图形时,纹理数据的传输速度需要200Mb/s以上,但PCI总线最高数据传输速度仅为133Mb/s,因而成为系统

的主要瓶颈。

为了解决3D图形数据的传输问题，主要的微机生产厂商联合推出了AGP图形接口。AGP在主内存与显卡之间提供了一条直接的通道，使得3D图形数据可以不经过PCI总线而直接送入显示子系统。这样就突破了由于PCI总线形成的系统瓶颈，从而实现了以相对低价格来达到高性能3D图形的描绘功能。因此，推出AGP接口的主要目的就是大幅提高微型机的3D图形处理能力，或者说AGP是用于加速图形显示的一个专用总线接口。

(2) AGP的性能特点。AGP以66MHz PCI Rev2.1规范为基础，在此基础上扩充了以下主要功能。

① 数据读写采用流水线操作，从而减少了内存等待时间，提高了数据传输速度。

② 具有2X、4X、8X的数据传输频率。AGP使用了32位数据总线和多时钟技术的66MHz时钟。因为时钟频率提高到了66MHz，所以带宽是PCI总线的两倍，达到了266Mb/s。随后很快AGP 2X问世，通过每周期传送两次32位数据将带宽提高到了533Mb/s。以后又出现了每时钟周期处理4个32位数据的AGP 4X模式，使AGP总线传输带宽突破了1Gb/s，达到了1066Mb/s。最新的8X模式使AGP带宽甚至可达2133Mb/s。

③ 直接内存执行DIME。AGP允许3D纹理数据直接存入系统内存，从而让出帧缓冲区和带宽供其他功能使用。这种允许显卡直接操作主存的技术称为DIME(Direct Memory Execute)。要说明的是，虽然AGP把纹理数据存入主存，但并没有完全取代显卡的显示缓存，AGP主存只是对缓存的扩大和补充。

④ 地址信号与数据信号分离。

⑤ 并行操作。在CPU访问系统RAM的同时允许AGP显卡访问AGP内存，显卡可以独享AGP总线带宽，从而进一步提高了系统性能。

5) PCI Express总线

PCI Express是新一代的总线接口，2002年由Intel公司联合AMD、DELL、IBM等在内的多家业界主导公司提出并完成。它采用点对点串行连接，比起PCI总线的共享并行架构，每个设备都有自己的专用连接，不需要向整个总线请求带宽，而且可以把数据传输率提高到一个很高的频率，达到PCI所不能提供的高带宽。

PCI Express总线接口从性能上主要具有以下特点。

(1) PCI Express在技术上允许实现X1、X2、X4、X8、X12、X16和X32通道规格，但目前来讲，PCI Express X1和PCI Express X16是PCI Express的主流规格。

(2) PCI Express X1支持双向数据传输，每向数据传输带宽为250Mb/s，可以满足主流声效芯片、网卡芯片和存储设备对数据传输带宽的需求，但是无法满足图形芯片对数据传输带宽的需求。

(3) PCI Express X16专为显卡设计，用于取代AGP接口以提高图形和视频信号的传输率，它也支持双向数据传输，能够提供5Gb/s的带宽，除去编码上的损耗，仍能够提供约4Gb/s的实际带宽，远远超过了AGP 8X的2.1Gb/s的带宽。

(4) 除去提供极高数据传输带宽之外，和ISA、PCI、AGP总线不同的另一点是PCI

Express 采用串行方式传输数据,因此其每个针脚可以获得比传统 I/O 标准更多的带宽,这样就降低了 PCI Express 设备的生产成本和体积。

(5) PCI Express 支持高阶电源管理、支持热插拔、支持数据同步传输,为优先传输数据进行带宽优化。

(6) 在软件层面上,PCI Express 兼容目前的 PCI 技术和设备,支持 PCI 设备和内存模组的初始化。也就是说目前的驱动程序、操作系统无须推倒重来,就可以支持 PCI Express 设备。

总之,PCI Express 是新一代的系统总线标准,其较高的数据传输性能能大幅提高中央处理器(CPU)和图形处理器(GPU)之间的带宽。

2. 外设总线

外部设备总线用于实现计算机主机和外部设备之间的连接,它与传统外设接口有很大的区别。传统外设接口是专用的,通常只能连接某一特定类型的设备,而且大多数情况下只能连接一个设备;外部设备总线是通用的,可连接不同的外部设备,并且允许在一个总线上连接很多设备。

常见的外部设备总线有 USB(Universal Serial Bus)和 IEEE 1394(又称 FireWire)。限于篇幅,下面仅简要介绍 USB 总线的特点及主要技术指标。

1) USB 总线

USB 是由 Compaq、DEC、IBM、Intel、Microsoft 和 NEC 等多家美国和日本公司共同开发的一种新的外设连接技术,其目的是为用户提供一种独立于主机系统,并在整个计算机系统结构中保持一致的,具有可共享、可扩充、使用方便等特性的串行总线。USB 具有以下一些特点。

(1) 易使用,主要表现在以下方面。

① 适合多种设备。USB 是一种通用接口,可适用于多种外设,即无须为每个外设准备不同的接口和协议,一种接口就能满足多种外设。

② 自动配置,即插即用(PnP)。当用户连接 USB 外设到一个正在运行的系统时,Windows 能自动检测外设,加载合适的驱动程序。

③ 无须用户设定。USB 不需要用户进行初始设置,例如端口地址和中断请求(IRQ)线等,这给使用带来了很大的方便。

④ 节省硬件资源。PC 上可供使用的 IRQ 线是一种宝贵的稀缺资源,无法给新的外设分配 IRQ 常常是使用 USB 的原因之一。如果外设都尽可能地使用 USB,就可使 IRQ 线空闲出来供那些必须使用 IRQ 的外设使用。对 USB 来说,它只需要若干个端口地址和一根 IRQ,而挂接到 USB 上的外设不需要其他任何资源。对比之下,每个非 USB 外设都要求有自己的端口地址,通常还要一根 IRQ 线,有时还要有一个扩展槽(如 Modem 卡)。

⑤ 易于连接。有了 USB,就不需要再打开计算机的机箱去为每个外设增加扩展卡。USB 的连接器和电缆都有确定的规格,即使没有经验的用户也不会接错。一个普通的 PC 有 2~6 个 USB 端口,如果需要的话,还可以通过连接一个 USB 集线器来扩展端口的数量。集线器可以提供最多 7 个端口来连接更多的外设或集线器。一个 USB 可支持多

达 127 个物理外设。

⑥ 可热插拔。不管系统和外设是否开机,都可以在任何时候连接和断开外设,且不会造成损坏。当外设连接到 PC 上时,操作系统会自动检测到并准备使用。

⑦ 不需另备电源。USB 接口自带了电源线和地线,可以提供+5V 的电源供应。一个外设如果需要中等功率的电源供应(最多 500mA),则它完全可以从总线得到电源供应而不需要使用外置电源。

(2) 速度较快。一个全速 USB 1.1 接口可以 12Mb/s 的速度进行通信。实际数据传输速率比这个数值要低一些,这是因为所有外设都共用总线,导致总线除传输数据外,还必须携带状态、控制和错误检测信号。如果这还不够快,USB 2.0 规范将允许以 480Mb/s 传输数据。这使得 USB 对打印机和其他需要快速传递大容量数据的外设更具吸引力。USB 也支持 1.5Mb/s 的低速传输。低速外设通常很便宜,而且它们的电缆可以更灵活(如鼠标),因为电缆不需要屏蔽。

(3) 可靠性高。USB 的可靠性来自于硬件设计和数据传输协议两方面。USB 驱动器、接收器和电缆的硬件规范消除了大多数可能引起数据错误的噪声。此外,USB 协议采用了差错控制/缺陷发现机制,当检测到错误时能通知发送方重新发送前面的数据。检测、通知和重发都由硬件来完成,不需要任何软件的介入。

(4) 低成本。虽然 USB 比以前的接口更复杂,但它的组件和电缆并不昂贵。带有 USB 接口的设备与带有相同功能的老式接口的设备所需的费用几乎是相同的,甚至更低。对成本非常低的外设来说,可以选择低速传输以降低对硬件的要求,使成本控制在合理的范围内。

(5) 低功耗。当 USB 外设不被使用时,省电电路和代码会自动关闭它的电源,但仍然能够在需要的时候做出反应。降低电源消耗除了可带来保护环境的好处之外,这个特征对于电源供应非常敏感的笔记本电脑尤其有用。

2) 主要技术指标

到目前为止,USB 已有 3 种版本:USB 1.1、USB 2.0 和 USB 3.0。这 3 种版本的 USB 均采用一条 4 芯的电缆连接主机和 USB 设备。连接电缆除提供信号线外,还向 USB 设备提供了电源。随着技术的发展,USB 的技术指标也在不断变化中,最新的 USB 3.0 的最大传输带宽为 5.0Gb/s。

2.5.4 8088 系统总线

在对总线系统有了整体了解的基础上,本节将简单介绍 8088 的系统总线结构。

1. 最小模式下的系统总线

在最小模式下(MN/$\overline{MX}$引脚接高电平),CPU 仅支持由少量设备组成的单处理器系统而不支持多处理器结构。这种模式下的 8088 系统总线构成如图 2-38 所示。图中,系统总线的 20 条地址线用 3 片 8282(或 74LS373)锁存器构成。8 条双向的数据总线通过 1 片 8286(或 74LS245)双向总线驱动器连接到外部数据总线。CPU 本身产生全部总线控制信号(DT/$\overline{R}$、$\overline{DEN}$、ALE 和 IO/$\overline{M}$)和命令输出信号($\overline{WR}$、$\overline{RD}$或 INTA),并提供请求访

问总线的控制信号(HOLD/HLDA),该信号与总线主设备控制器(如 Intel 8237 和 8257DMA 控制器)兼容。这样就实现了最小模式下的系统总线。在实际系统中,还应考虑以下两个问题。

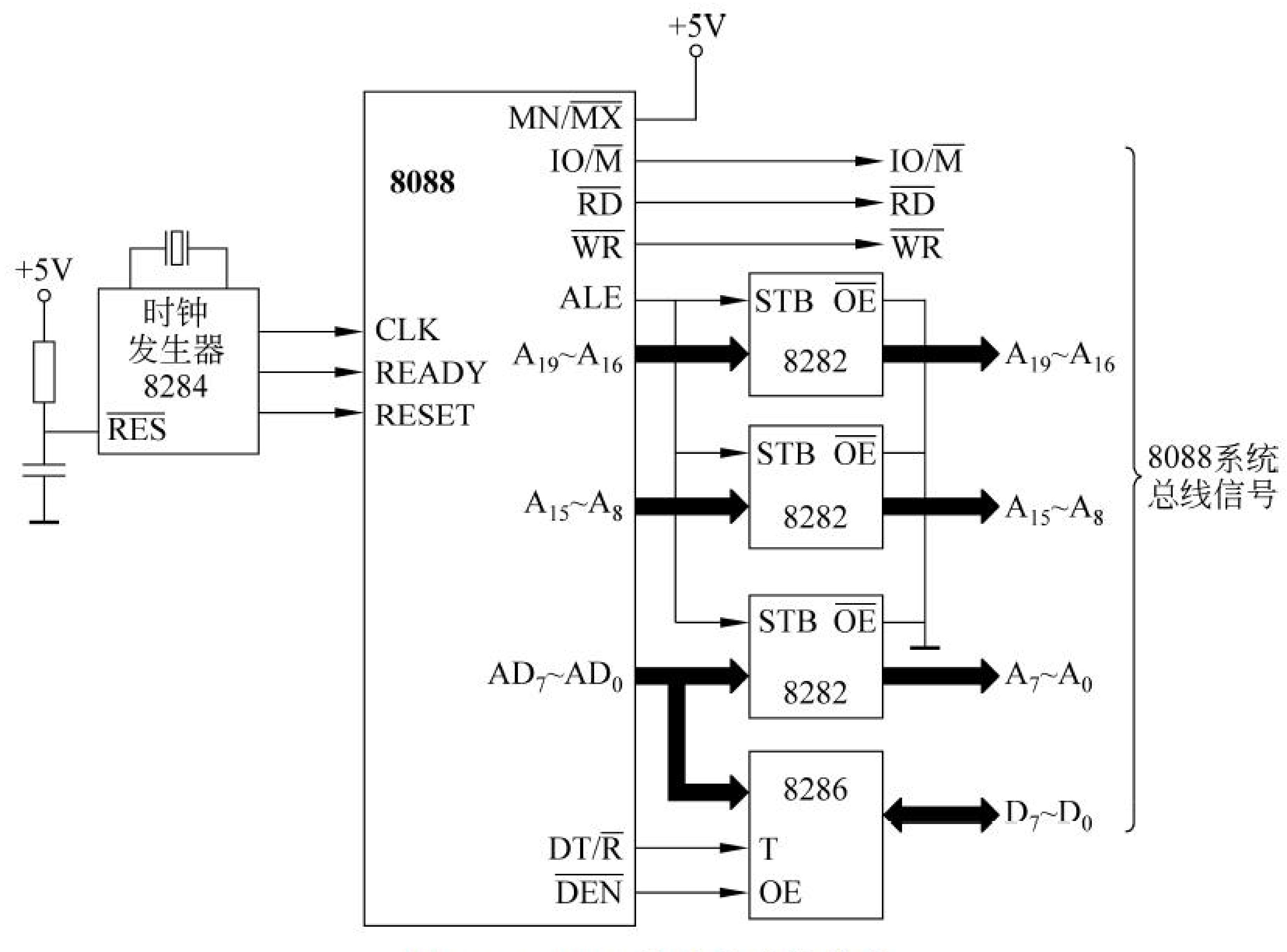


图 2-38　8088 的最小总线模式

① 系统总线的控制信号是 8088 CPU 直接产生的。若 8088 CPU 驱动能力不够,可以加上总线驱动器 74LS244 进行驱动。

② 按此构成的系统总线尚不能进行 DMA 传送,因为未对系统总线形成器件(8282、8286)做进一步控制。

2. 最大模式下的系统总线

在最大模式(MN/$\overline{MX}$引脚接低电平)下,增添一个 8288 总线控制器就使 CPU 能支持系统总线上的多个处理器。图 2-39 为最大模式的系统组成框图。在最大模式下,由总线控制器提供所有总线控制信号和命令信号。CPU 的部分引脚进行了重新定义以支持多处理器工作方式。8288 总线控制器利用 CPU 输出的$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$状态信号来产生总线周期所需的全部控制和命令信号。S_2、S_1、S_0 状态信号的定义可参见附录 B(B.2)。

在图 2-39 中,8282 和 8286 也可分别用 74LS373 和 74LS245 代替。在此图中同样没有考虑在系统总线上实现 DMA 传送的问题。下面提到的在 PC/XT 系统总线上所采用的 DMA 传送方法是一种解决方案,总的原则就是:在进行 DMA 传送时,一定要保证总线形成电路所有输出信号都呈现高阻状态,即放弃对系统总线的控制。

当系统总线形成之后,内存及各种接口就可以直接与系统总线相连接,从而构成所需的微型机系统。鉴于在后面章节中要经常用到 8088 最大模式下的总线信号,希望读者能牢固掌握以下系统总线信号的作用及它们互相之间的定时关系。

(1) 地址信号线:A_0～A_{19}。

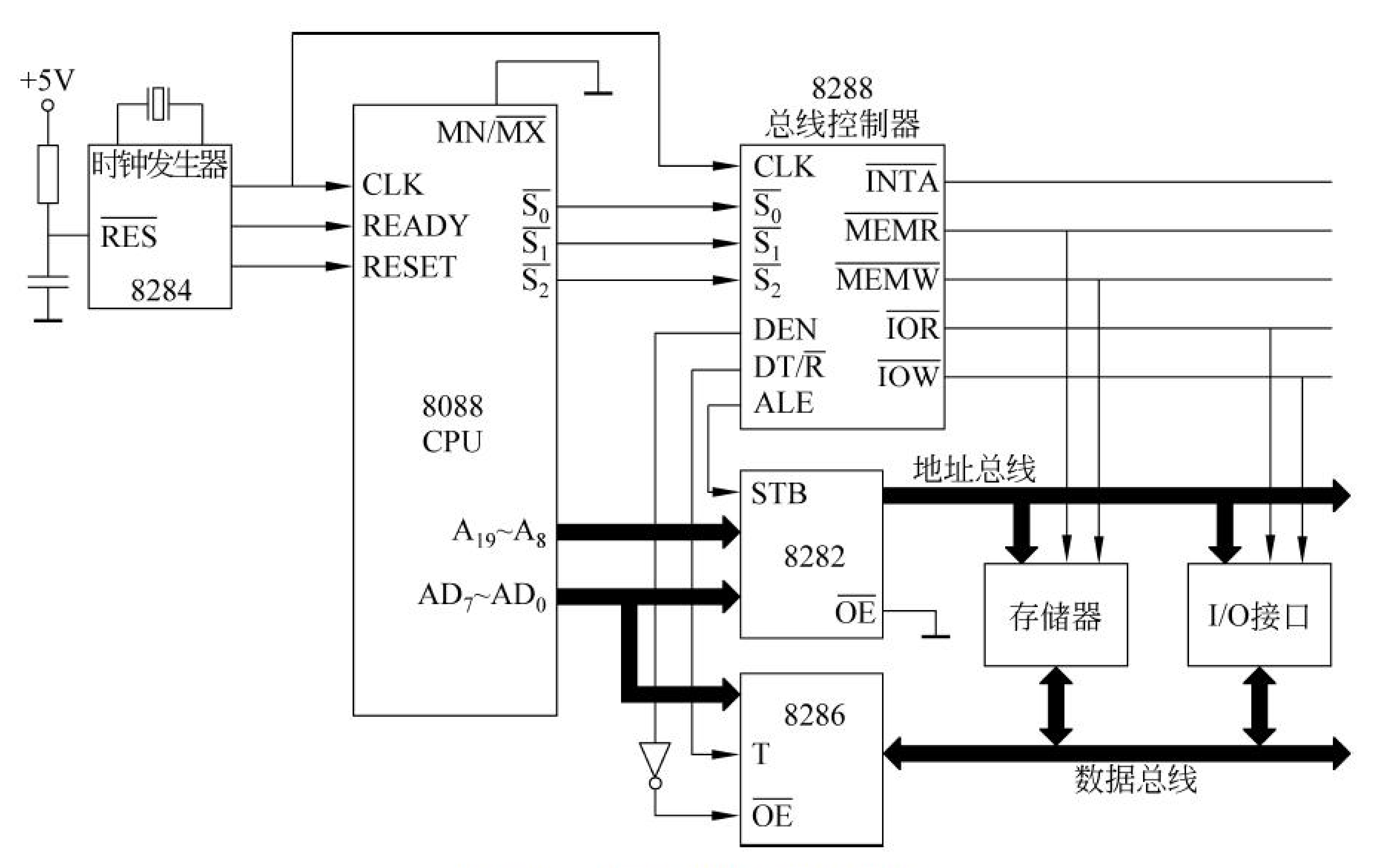


图 2-39　8088 的最大总线模式

(2) 数据信号线：$D_0 \sim D_7$。

(3) 控制信号线：$\overline{MEMR}$、$\overline{MEMW}$(访问存储器时的读、写控制信号)；$\overline{IOR}$、$\overline{IOW}$(访问 I/O 端口时的读、写控制信号)。

在后面的章节中将直接采用系统总线信号来叙述问题，不再做出说明。

2.6　多核技术

自 1996 年美国斯坦福大学首次提出片上多处理器(CMP)思想和首个多核结构原型，到 2001 年 IBM 推出第一个商用多核处理器 Power4，再到 2005 年 Intel 和 AMD 多核处理器的大规模应用，最后到现在多核成为市场主流，多核处理器经历了多年的发展。在这个过程中，多核处理器的应用范围已覆盖了多媒体计算、嵌入式设备、个人计算机、商用服务器和高性能计算机等众多领域，多核技术及其相关研究的发展非常迅速。

2.6.1　什么是多核技术

1. 多核的概念

多核处理器将多个完全功能的核心集成在同一个芯片内，整个芯片作为一个统一的结构对外提供服务。首先，多核处理器通过集成多个处理核心，使得整个处理器可同时执行的线程数或任务数是单处理器的数倍，极大地提升了处理器的并行性能；其次，多个核集成在片内，极大地缩短了核间的互连线，核间通信延迟变低，提高了通信效率，数据传输带宽也得到了提高；再次，多核结构有效共享资源，片上资源的利用率得到了提高，功耗也

随着器件的减少得到了降低；最后，多核结构简单，易于优化设计，扩展性强。这些优势最终推动了多核的发展并逐渐取代单核处理器成为主流。

双核 PC 时代在 2005 年 4 月正式开始，当时 Intel 发布了至尊版奔腾 840 处理器，是一款主频 3.2GHz 的 90 纳米芯片，紧随其后的就是奔腾 D 800 系列 CPU。而英特尔的第二代双核处理器奔腾 D 900 系列在 2006 年初发布，开始将 Intel 的制程工艺全面转向 65 纳米。表 2-4 给出了 Intel 系列处理器结构的发展流程，可以看出，随着时间的推移，多核处理器迅速进入主流，总体上处理器核心数量也处于越来越多的趋势。

表 2-4 Intel 系列处理器结构特征

结构代号	年份	处理器核数	结构代号	年份	处理器核数
P5	1993	1	Westmere	2010	2～10
P6	1995	1	Sandy bridge	2011	1～8
Core	2006	1～4	Ivy bridge	2012	2～15
Penryn	2007	1～4	Haswell	2013	4～18
Nehalem	2008	2～8			

目前的研究一般认为，多核处理器相比相同工艺、相同面积的单核处理器具有如下优势：

(1) 逻辑简单：相对超标量微处理器结构和超长指令字结构而言，单芯片多处理器结构的控制逻辑复杂性要明显低很多。相应的单芯片多处理器的硬件实现必然要简单得多。

(2) 高主频：芯片多处理器核心结构的控制逻辑相对简单，包含极少的全局信号，因此线延迟对其影响较小，因此，在同等工艺条件下，多核处理器的硬件实现可以取得比超标量微处理器和超长指令字微处理器更高的工作频率。

(3) 低通信延迟：由于多个处理器核心集成在一块芯片上，且采用共享 Cache 或者内存的方式，多线程的通信延迟会明显降低，这样也对存储系统提出了更高的要求。

(4) 低功耗：调节电压/频率、负载优化分布等，可有效降低功耗。

(5) 设计和验证周期短：微处理器厂商一般采用现有的成熟单核处理器作为处理器核心，从而可缩短设计和验证周期，节省研发成本。

虽然在总体性能和能源效率方面上多核具有明显优势，但是从目前多核的技术和人们对于其应用能力上看，还有两方面的潜在问题：

(1) 为了达到总体性能和能源的有效性，在同一工艺条件下，每个核心在芯片上所占的面积实际上较小，意味着每个核心比相应的单核处理器要简单，从而计算能力相对较弱。对于那些本质上必须串行执行的程序来讲，由于很难利用到多个核心，因此它们在多核情况下可能会运行得更慢。一般来讲，不能简单地期望 N 核处理器能够达到 N 倍的性能。

(2) 当核心数目增多时，虽然理论上可以通过并行处理得到性能提升，但是目前人们并没有完全清楚如何将各种类型的应用有效分布到各个并行处理单元上协同工作。另外，从体系结构的角度来讲，多个核心如何能有效地互联通信，如何有效地共享缓存资源，以及如何能够在有限的片外管脚数目上达到多个核心总体需求的 I/O 带宽等问题都还具有很大的挑战性。

2. 多核的关键技术

多核处理器结构不仅有性能潜力大、集成度高、并行度高、结构简单和设计验证方便等诸多优势，而且它还能继承传统单处理器研究中的某些成果，例如超线程、宽发射指令、降压低功耗技术等。多核处理器毕竟是一种新的结构，在多核结构设计和应用开发中也出现了以前未曾遇到的新问题，这些问题给多核处理器的未来提出了挑战。

1）核心结构

对于核心结构的选择，目前多核处理器的核心结构主要有同构和异构两种。同构与异构是多核处理器主要的两种结构形态。顾名思义，同构多核处理器是指处理器芯片内部的所有核心其结构是完全相同的，各个核心的地位也是等同的。目前的同构多核处理器大多数由通用的处理器核心组成，每个处理器核心可以独立执行任务，与通用单核处理器结构相近。而异构多核则是将结构、功能、功耗、运算性能各不相同的多个核心集成在芯片上，并通过任务分工和划分将不同的任务分配给不同的核心，让每个核心处理自己擅长的任务。

除了同构和异构的区分之外，核心本身的结构还关系到整个芯片的面积、功耗和性能。怎样继承和发展传统处理器的成果，直接影响多核的性能和实现周期。核所用的指令系统对系统的实现也是很重要的。多核之间采用相同的指令系统还是不同的指令系统，能否运行操作系统等，也将是研究的内容之一。

2）程序执行模型

处理器设计的首要问题是选择程序执行模型。程序执行模型的适用性决定多核处理器能否以最低的代价提供最高的性能。程序执行模型是编译器设计人员与系统实现人员之间的接口。编译器设计人员决定如何将一种高级语言程序按一种程序执行模型转换成一种目标机器语言程序；系统实现人员则决定该程序执行模型在具体目标机器上的有效实现。当目标机器是多核体系结构时，产生的问题是：多核体系结构如何支持重要的程序执行模型？是否有其他的程序执行模型更适于多核的体系结构？这些程序执行模型能在多大程度上满足应用的需要并为用户所接受？

3）多级 Cache 设计与一致性问题

处理器和主存间的速度差距对 CMP 来说是个突出的矛盾，因此必须使用多级 Cache 来缓解。目前有共享一级 Cache 的 CMP、共享二级 Cache 的 CMP 以及共享主存的 CMP。通常，CMP 采用共享二级 Cache 的 CMP 结构，即每个处理器核心拥有私有的一级 Cache，且所有处理器核心共享二级 Cache。Cache 自身的体系结构设计也直接关系到系统整体性能，但是在 CMP 结构中，共享 Cache 或独有 Cache 孰优孰劣，需不需要在一块芯片上建立多级 Cache，以及建立几级 Cache 等，由于对整个芯片的尺寸、功耗、布局、性能以及运行效率等都有很大的影响，因而这些都是需要认真研究和探讨的问题。另一方面，多级 Cache 又引发了一致性问题，采用何种 Cache 一致性模型和机制都将对 CMP 整体性能产生重要影响。

4）核间通信技术

多核芯片上的多个核心虽然各自执行自己的代码，但是不同核心间可能需要进行数

据的共享和同步，因此片上通信结构的性能将直接影响处理器的性能。高效的通信机制是CMP处理器高性能的重要保障，目前比较主流的片上高效通信机制有两种，一种是基于总线共享的Cache结构，一种是基于片上的互连结构。总线共享结构是指片上核心、输入输出端口以及存储器通过共享二级或三级Cache，或者通过连接核心的总线进行通信。总线结构的长处是较为简单，易于设计实现，当前多数双核和四核处理器基本上都采用了该结构，但缺点是总线结构可扩展性较差，适用于核心数较少的情况。比较典型的总线共享结构处理器有Hydra、Intel的Core、IBM的Power4/5等。基于片上互连的结构是指每个CPU核心具有独立的处理单元和Cache，各个CPU核心通过交叉开关或片上网络等方式连接在一起。各个CPU核心间通过消息通信，例如AMD公司的Athlon x2双核处理器用交叉开关来控制核心与外部的通信。这种结构的优点是可扩展性好，数据带宽有保证；缺点是硬件结构复杂，且软件改动较大。也许这两者的竞争结果不是互相取代而是互相合作。例如在全局范围采用片上网络而局部采用总线方式，来达到性能与复杂性的平衡。

5）总线设计

传统微处理器中，Cache不命中或访存事件都会对CPU的执行效率产生负面影响，而总线接口单元(BIU)的工作效率决定此影响的程度。当多个CPU核心同时要求访问内存或多个CPU核心内私有Cache同时出现Cache不命中事件时，BIU对这多个访问请求的仲裁机制以及对外存储访问的转换机制的效率决定了CMP系统的整体性能。Intel的快速通道互联(Quick Path Interconnect，QPI)总线技术，更大程度发掘了多核处理器的实力。

6）操作系统设计

对于多核CPU，优化操作系统任务调度算法是保证效率的关键。一般任务调度算法有全局队列调度和局部队列调度。前者是指操作系统维护一个全局的任务等待队列，当系统中有一个CPU核心空闲时，操作系统就从全局任务等待队列中选取就绪任务开始在此核心上执行。这种方法的优点是CPU核心利用率较高。后者是指操作系统为每个CPU内核维护一个局部的任务等待队列，当系统中有一个CPU内核空闲时，便从该核心的任务等待队列中选取恰当的任务执行。这种方法的优点是任务基本上无须在多个CPU核心间切换，有利于提高CPU核心局部Cache命中率。目前多数多核CPU操作系统采用的是基于全局队列的任务调度算法。多核的中断处理和单核有很大不同。多核的各处理器之间需要通过中断方式进行通信，所以多个处理器之间的本地中断控制器和负责仲裁各核之间中断分配的全局中断控制器也需要封装在芯片内部。另外，多核CPU是一个多任务系统，由于不同任务会竞争共享资源，因此需要系统提供同步与互斥机制。而传统的用于单核的解决机制并不能满足多核，需要利用硬件提供的“读—修改—写”的原子操作或其他同步互斥机制来保证。

7）低功耗设计

半导体工艺的迅速发展使微处理器的集成度越来越高，同时处理器表面温度也变得越来越高并呈指数级增长，每三年处理器的功耗密度就能翻一番。目前，低功耗和热优化设计已经成为微处理器研究中的核心问题。CMP的多核心结构决定了其相关的功耗研

究是一个至关重要的课题。低功耗设计是一个多层次问题,需要同时在操作系统级、算法级、结构级、电路级等多个层次上进行研究。每个层次的低功耗设计方法实现的效果不同——抽象层次越高,功耗和温度降低的效果越明显。

3. 多核的发展趋势

随着操作系统及应用软件对多核处理器的进一步支持及优化、芯片制造工艺的成熟、AMD 及 Intel 为代表的低功耗技术的发展、芯片级虚拟化技术的成熟等诸多因素,服务器处理器多核化趋势的进一步彰显,多核技术将成为服务器技术的重要技术支点。

应用需求的不断提高是计算机发展的根本动力。如目前的服务器应用、要求高的吞吐率和在多处理器上的多线程应用、Internet 的应用、P2P 和普适计算的应用都促进了计算机性能的不断提升,多核技术已经成为服务器技术的重要技术支点。大型企业的 ERP、CRM 等复杂应用,科学计算、政府的大型数据库管理系统、数字医疗领域、电信、金融等都需要高性能计算,多核技术可以满足这些应用的需求。

多核处理器的出现,对计算机体系结构的发展来讲有着深远的影响。在未来的一段时间之内,多核处理器将在处理器市场上占有统治地位。如何充分利用多核处理器的性能,更好地发挥出多核的优势,让每个核能够同时处理任务,提高系统利用率,不仅需要硬件上的资源重复,多核互联,还需要更好地分配任务。分配能够使多个核同时工作,不互相争夺共享资源的任务,是当前软件工作者和硬件工作者协同工作的重点所在。

多核处理器产生的直接原因是替代单处理器,解决微处理器的发展瓶颈,但发展多核的深层次原因还是为了满足人类社会对计算性能的无止境需求,而且这种压力还会持续下去。阻碍多核性能向更高水平发展的问题很多,可真正束缚多核发展的是低功耗和应用开发两个问题。由于现有的多核结构设计方法和技术还不能有效地处理好这两个问题,因此有必要在原有技术基础上探索新的思路和方法。

(1) 多核上将集成更多结构简单、低功耗的核心。为了满足性能需求,通过集成更多核心来提高性能是必然选择,但是核心的结构也必须考虑,因为如果核心结构过于复杂,那么随着核心数量的增多,不仅不能提升性能,还会带来线延迟增加和功耗变大等问题。

(2) 异构多核是一个重要的方向。异构组织方式比同构的多核处理器执行任务更有效率,实现了资源的最佳化配置,而且降低了整体功耗。

(3) 多核上应用可重构技术。大规模高性能可编程器件的出现,推动了现场可编程门阵列(Field Programmable Gate Arrays,FPGA) 技术的发展。在芯片上应用 FPGA 技术有高灵活性、高可靠性、高性能、低能耗和低成本多种优势。微处理器设计人员注意到了这种优势,并将 FPGA 等可重构技术应用到多核结构上,使其具备可重构性和可编程性。

目前,多核处理器的推广还受到一定程度的限制,如一些桌面应用尚不支持多线程,多核处理器价格相对偏高,应用开发工具不成熟等。随着应用需求的扩大和技术的不断进步,多核必将展示出其强大的性能优势。无论是移动与嵌入式应用、桌面应用还是服务器应用,多核处理器的高性能、低功耗的特性在满足广大客户对性能不断追求的同时也极大提升了性能功耗比,为广大用户提供了更好的选择。随着多核生态系统及制造工艺的

进一步成熟,多核技术的主流化趋势已势不可挡。

4. 多核技术的应用场景

越来越多的用户在使用过程中都会涉及到多任务应用环境,日常应用中非常典型的有两种应用模式。

一种应用模式是:如果一个程序采用了线程级并行编程,那么这个程序在运行时可以把并行的线程同时交付给两个核心分别处理,因而程序运行速度得到了极大提高。这类程序有的是为多路工作站或服务器设计的专业程序,例如专业图像处理程序、非线性视频编辑程序、动画制作程序或科学计算程序等。对于这类程序,两个物理核心和两个处理器基本上是等价的,因此这些程序往往可以不作任何改动就直接运行在双核计算机上。

还有一些更常见的日常应用程序,例如 Office、IE 等,同样也是采用线程级并行编程,可以在运行时同时调用多个线程协同工作,所以在多核处理器上的运行速度也会得到较大提升。例如,打开 IE 浏览器上网。看似简单的一个操作,实际上浏览器进程会调用代码解析、Flash 播放、多媒体播放、Java、脚本解析等一系列线程,这些线程可以并行地被多核处理器处理,因而运行速度大大加快。由此可见,对于已经采用并行编程的软件,不管是专业软件还是日常应用软件,在多核处理器上的运行速度都会大幅提高。

日常应用中的另一种模式是:同时运行多个程序。许多程序没有采用并行编程,例如一些文件压缩软件、部分游戏软件等。对于这些单线程的程序,单独运行在多核处理器上与单独运行在同样参数的单核处理器上没有明显的差别。但是,由于日常使用的最最基本的程序——操作系统——是支持并行处理的,所以,当在多核处理器上同时运行多个单线程程序的时候,操作系统会把多个程序的指令分别发送给多个核心,从而使得同时完成多个程序的速度大为加快。

2.6.2 多核与多处理器

多核技术能够使计算机在只有一个处理器的情况下实现任务的并行处理,而在多核技术蓬勃发展以前,并行计算任务必须使用多个独立的处理器进行协同计算。

多核与多处理器架构的主要区别在于:

(1) 核心间通信速度:多核是指一个处理器芯片有多个处理器核心,它们之间通过 CPU 内部总线进行通信;而多处理器架构是由多个相同或者不同的独立完整的 CPU 通过通信通道连接,可共享也可独立拥有存储器、外设,多个处理器之间的通信是通过主板上的总线甚至百兆、千兆网线或光纤进行的;两者核心间的通信速度有着数量级的差别。

(2) 开发难度:多核处理器采用与单 CPU 相同的硬件架构,用户在提升计算能力的同时无须进行任何硬件上的改变;而多处理器系统目前常见于分布式系统中,必然要面临大量的数据一致性、主从关系控制、可靠性保障问题,开发难度较大。

(3) 使用场合:多处理器架构一般不用于普通的消费级市场,多用于服务器集群、云计算平台等场合。这些场合一般计算量需求很大,对速度不过于敏感;而且多处理器架构更简单、清晰,可以用消费级产品简单进行数量堆叠,处理器数量动辄以万为单位,单位成

本较低。而多核则相对较适合普通桌面应用，单位成本较高，核心数目较少，为了控制成本，目前普通消费级产品最多也就是16核左右。

习　　题

2.1　微处理器主要由哪几部分构成？

2.2　什么是多核处理器？

2.3　说明8088 CPU中EU和BIU的主要功能。在执行指令时，EU能直接访问存储器吗？

2.4　8088 CPU工作在最小模式时：

(1) 当CPU访问存储器时，要利用哪些信号？

(2) 当CPU进行I/O操作时，要利用哪些信号？

(3) 当HOLD有效并得到响应时，CPU的哪些信号置高阻？

2.5　总线周期中，何时需要插入 T_W 等待周期？插入 T_W 周期的个数取决于什么因素？

2.6　若8088工作在单CPU方式下，在表2-5中填入不同操作时各控制信号的状态。

表2-5　控制信号的状态

操作	$IO/\overline{M}$	$DT/\overline{R}$	$\overline{DEN}$	$\overline{RD}$	$\overline{WR}$
读存储器					
写存储器					
读I/O接口					
写I/O接口					

2.7　在8086/8088 CPU中，标志寄存器包含哪些标志位？各位为0(为1)分别表示什么含义？

2.8　8086/8088 CPU中，有哪些通用寄存器和专用寄存器？说明它们的作用。

2.9　8086/8088系统中，存储器为什么要分段？一个段最大为多少字节？最小为多少字节？

2.10　在8088 CPU中，物理地址和逻辑地址是指什么？已知逻辑地址为1F00:38A0H，如何计算出其对应的物理地址？若已知物理地址，其逻辑地址唯一吗？

2.11　若CS＝8000H，则当前代码段可寻址的存储空间的范围是多少？

2.12　8086/8088 CPU在最小模式下的系统构成至少应包括哪些基本部分(器件)？

2.13　在图2-34中，若设备接口0和设备接口1同时申请总线，哪一个设备接口将最先获得总线控制权？为什么？

2.14　在南北桥结构的80x86系统中，PCI总线是通过什么电路与CPU总线相连的？ISA总线呢？

2.15　现代微机系统中，总线可分为哪些类型？主要有哪些常用系统总线和外设总线

标准？

2.16 80386 CPU 包含哪些寄存器？各有什么主要用途？

2.17 什么是实地址模式？什么是保护模式？它们的特点是什么？

2.18 80386 访问存储器有哪两种方式？各提供多大的地址空间？

2.19 如果 GDT 寄存器值为 0013000000FFH，装入 LDTR 的选择符为 0040H，试问装入缓存 LDT 描述符的起始地址是多少？

2.20 页转换产生的线性地址的三部分各是什么？

2.21 选择符 022416H 装入了数据段寄存器，该值指向局部描述符表中从地址 00100220H 开始的段描述符。如果该描述符的内容为

(00100220H) = 10H，(00100221H) = 22H

(00100222H) = 00H，(00100223H) = 10H

(00100224H) = 1CH，(00100225H) = 80H

(00100226H) = 01H，(00100227H) = 01H

则段基址和段界限各为多少？

2.22 Pentium 4 的基本程序执行环境包含了哪些寄存器？

2.23 什么是多核技术？多核和多处理器的主要区别是什么？

第3章 8086/8088 指令系统

引言：

每一系列的处理器都有自己的指令系统，可以说，指令系统功能的强弱大体上决定了计算机硬件系统功能的高低。本章以 8086/8088 CPU 指令系统为基础，介绍指令的一般概念和执行过程、CISC 和 RISC 指令的概念、寻址方式以及不同类型指令的功能。

教学目的：

(1) 了解指令的一般概念、指令的基本格式及指令的执行过程；

(2) 熟悉指令对操作数的各种寻址方式；

(3) 深入理解 8086 指令系统全部六大类指令的功能，包括指令操作码的含义、指令对操作数的要求和指令的执行结果。

3.1 概　　述

控制计算机完成指定操作并能够被计算机所识别的命令称为指令。一台计算机能够识别的所有指令的集合称为该机的指令系统。不同的计算机(或者说不同的微处理器)具有各自不同的指令系统。指令系统定义了计算机硬件所能完成的基本操作，其功能的强弱在一定程度上决定了硬件系统性能的高低。

Intel 8088/8086 CPU 指令系统也是 Intel 80x86 系列 CPU 的基本指令系统。由于 8086 和 8088 的指令系统完全相同，为叙述方便，以下统称为 8086 指令系统。

8086 CPU 与其上一代的 8 位 CPU 如 8080、8085 相比，其指令系统在指令的数量上、功能上、寻址方式的多样性上以及处理数据的能力上都有了很大的提高。例如，8086 不仅有加减法指令，还可用一条指令完成乘法或除法运算。此外它还增加了中断指令及串操作指令等。

8086/8088 CPU 的指令系统共包含 92 种基本指令，按照功能可将它们分为六大类：数据传送类、算术运算类、逻辑运算和移位、串操作、控制转移类、处理器控制。

为使读者对 8086/8088 指令系统有一个粗略的概念，表 3-1 列出了上述六大类指令中常用指令的助记符。更详细的内容将在 3.3 节中介绍。

表 3-1　8086/8088 CPU 常用指令一览表

指令类型		助　记　符
数据传送	一般数据传送	MOV、PUSH、POP、XCHG、XLAT、CBW、CWD
	输入输出指令	IN、OUT
	地址传送指令	LEA、LDS、LES
	标志传送指令	LAHF、SAHF、PUSHF、POPF
算术运算	加法指令	ADD、ADC、INC
	减法指令	SUB、SBB、DEC、NEG、CMP
	乘法指令	MUL、IMUL
	除法指令	DIV、IDIV
	十进制调整指令	DAA、AAA、DAS、AAS、AAM、AAD
逻辑运算和移位指令		AND、OR、NOT、XOR、TEST、SHL、SAL、SHR、SAR、ROL、ROR、RCL、RCR
串操作		MOVS、CMPS、SCAS、LODS、STOS
控制转移指令		JMP、CALL、RET、LOOP、LOOPE、LOOPNE、INT、INTO、IRET 各类条件转移指令
处理器控制指令		见表 3-5

3.1.1　指令的基本构成

1. 指令的一般格式

一条指令通常由两个部分组成，如图 3-1 所示。第一部分为操作码(或称指令码)，用便于记忆的助记符表示(一般是英文单词的缩写)，用于指出指令要进行何种操作，因此是指令中必须给出的内容。另一部分是指令操作的对象，称为操作数，可根据不同的情况显式地给出或隐含存在。

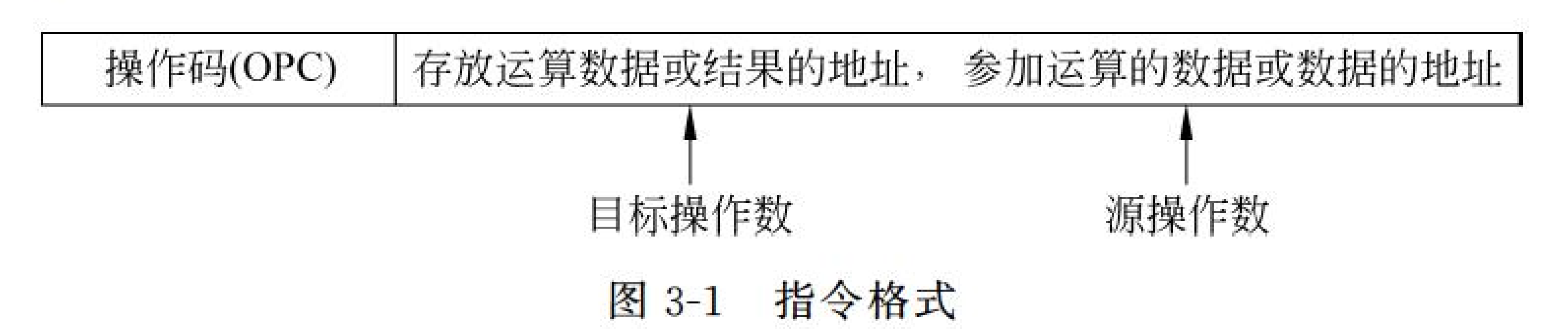


图 3-1　指令格式

指令的长度(所占的字节数)会影响指令的执行时间。8086 指令的长度在 1～7 个字节之间。操作码占用一个字节或两个字节。指令的长度主要决定于操作数的个数及所采用的寻址方式。在微处理器指令系统中，一条指令的操作数可以没有或有一个，但最多只能有两个。相应地，指令在格式上就有以下 3 种形式。

(1) 零操作数指令。指令在形式上只有操作码，操作数是隐含存在的。这类指令操作的对象通常为处理器本身。

(2) 单操作数指令。指令中仅给出一个操作数,另一个操作数隐含存在。

(3) 双操作数指令。格式如图 3-1 所示。

2. 指令中的操作数类型

8086 指令中的操作数主要有 3 种类型:立即数操作数、寄存器操作数和存储器操作数。

1) 立即数操作数

立即数是指具有固定数值的操作数,即常数,它不因指令的执行而发生变化。在 8086 系统中,立即数的字长可以是 1 字节或 2 字节;可以是无符号数或有符号数。要求数的取值范围必须符合相应字长数的规定,如果取值超出了规定的范围,就会发生错误。

在指令中,立即数操作数只能用作源操作数,而不能用作目标操作数。原因是立即数是一个常数,没有表示地址的含义。

2) 寄存器操作数

8086 CPU 的 8 个通用寄存器和 4 个段寄存器可以作为指令中的寄存器操作数,它们既可以作为源操作数,也可以用作目标操作数。

通用寄存器通常用来存放参加运算的数据或数据所在存储器单元的偏移地址。段寄存器用来存放当前操作数的段基地址。

仅有个别指令将标志寄存器 FLAGS 作为指令的操作数。

3) 存储器操作数

存储器操作数的含义是:参加运算的数据是存放在内存中的。由于 8086 指令系统中的操作数一般均为 8 位或 16 位字长,所以存储器操作数的字长也通常为字节或字,极个别的指令中有双字长的操作数。

存储器操作数在指令中既可作为源操作数,也可作为目标操作数。

第 2 章已经学习,能够唯一标识一个存储器单元的是它的物理地址,物理地址由段基地址和偏移地址两部分构成。所以,要寻找一个存储器操作数,必须首先确定操作数所在的逻辑段。一般情况下,若指令中没有明确指出操作数所在段,则 CPU 就采用默认的段寄存器来确定操作数的段基地址。各种存储器操作数所约定的默认段寄存器、段重设(即显式地指明段寄存器)所允许的段寄存器以及指令的有效地址所在寄存器请参见表 2-3。

存储器操作数的偏移地址(Efficient Address,EA)(也称有效地址)可以通过不同的寻址方式由指令给出。实际上,3.2 节中讲到的各种较复杂的寻址方式,大多都是针对存储器操作数的。

3.1.2 指令的执行时间

了解指令的执行时间,在有些时候是很重要的。例如在用软件产生定时或延时时,需要估算出一段程序的运行时间。另外,在某些实时控制要求较严或对程序运行时间要求较高的的场合,除需认真研究程序的算法外,对选择什么样的指令及采用什么样的寻址方

式也是很重要的。因为不同的指令在执行时间上有很大的差别，而不同的寻址方式其计算偏移地址所需时间也不同。由于指令的种类很多，要详细讨论各种指令的执行时间比较困难，这里只作一般的讨论。

一条指令的执行时间应包括取指令、取操作数、执行指令及传送结果几个部分，单位用时钟周期数表示。

不同指令的执行时间有较大的差别（见附录 C.1）。寄存器操作数占用的时间最短。存储器操作数的时间与采用的寻址方式有关，不同的寻址方式，计算偏移地址（EA）所需要的时间不同，其指令执行时间可能会相差很大。

在 3.1.1 节中讨论的 3 种类型的操作数中，寄存器操作数的指令执行速度最快，立即数操作数次之，存储器操作数指令的执行速度最慢。这是由于寄存器位于 CPU 的内部，执行寄存器操作数指令时，8086 的执行单元（EU）可以简捷地从 CPU 内部的寄存器中取得操作数，不需要访问内存，因此执行速度很快；立即数操作数作为指令的一部分，在取指时被 8086 总线接口单元（BIU）取出后存放在 BIU 的指令队列中，执行指令时也不需要访问内存，因而执行速度也比较快；而存储器操作数存放在内存单元中，为了取得操作数，首先要由总线接口单元计算出其所在单元的 20 位物理地址，然后再执行存储器的读写操作。所以相对前述两种操作数来说，指令的执行速度最慢。

以通用数据传送指令（MOV）为例，若 CPU 的时钟频率为 5MHz，即一个时钟周期为 0.2μs，则从寄存器到寄存器之间的传送指令的执行时间为

$$t=2\times0.2=0.4\mu s$$

立即数传送到寄存器的指令执行时间为

$$t=4\times0.2=0.8\mu s$$

而存储器到寄存器的字节传送，设存储器采用基址—变址寻址方式，则指令执行时间为

$$t=(8+EA)\times0.2=(8+8)\times0.2=3.2\mu s$$

3.1.3 CISC 和 RISC 指令系统

不同系列的 CPU 有不同的指令系统。目前，指令系统的设计有两个完全不同的方向。一个称为复杂指令系统计算机（Complex Instruction Set Computer，CISC），另一个是 20 世纪 80 年代新发展起来的、以简化指令功能为主要目的的精简指令系统计算机（Reduced Instruction Set Computer，RISC）。

1. CISC 指令

不同系列的 CPU 有不同的指令系统，每一种 CPU 都有属于它自己的指令系统。CPU 正是通过执行一系列特定的指令来满足应用程序的特定要求的。CISC 指令的设计目标是增强指令的功能，将一些原来用软件实现的、常用的功能变成用硬件的指令系统来实现。例如，在科学计算的应用程序中，经常要计算各种各样的函数，有些计算机系统就设置了一些常用的函数运算指令，用一条指令代替软件的一个子程序来完成函数计算。

随着超大规模集成电路(VLSI)技术的发展,计算机硬件的成本不断下降,而软件成本却不断地上升,操作系统的效率和微机的性能的进一步提高促使整个指令系统在功能上有以下的改进。

(1) 在指令系统中增加更多的指令和功能更强的复杂的指令,将使用频率高的指令串用一条新的指令去取代,将使用频率高的指令用硬件加快其执行。这样就使得程序的长度和执行时间都得以缩短。

(2) 增加对高级语言和编译程序支持的指令的功能,以减少编译时间,缩短目标程序的长度,进一步降低软件成本。

(3) 尽可能缩机器语言与高级语言的差距。众所周知,编译程序的作用就是把由高级语言编写的语句翻译成一个机器指令序列,如若机器指令与高级语言的语句相类似,编译程序的任务就简单多了。这样走到极端,就是将高级语言与机器语言合二为一,构成所谓的高级语言计算机。

(4) 增加对操作系统支持的指令,以实现对操作系统的优化。有些支持操作系统的指令属于特权指令,对一般用户不公开。这类指令中,有些指令的使用频率并不高,但如果没有它们的支持,操作系统将很难实现,如处理机转换、进程切换等方面所使用的指令。

为使新的微机与其前代机在软件上兼容,指令系统只能扩充,不能减少,从而使得微机的指令系统越来越复杂。如在 Pentium 微处理机指令系统内不仅继承下它的前辈机的所有指令,而且又增加了 Cache 的指令和诸如 8 字节比较和交换等指令,指令数达 300 余条。

复杂指令难以使用这是一个不争的事实。因为编译程序必须使每一条由高级语言编写的语句经编译后,满足所生成的指令代码的长度最小、指令执行的次数最少、适合流水线操作等诸多优化所生成指令的条件。所以使用复杂指令系统是一件并不轻松的工作,尤其是非计算机专业的人士。

CISC 也有许多优点,如指令经编译后生成的指令程序较小、执行起来较快、节省硬件资源、存取指令的次数少、占用较少的存储器等。

2. RISC 指令

从计算机诞生之日起,人们就在不断地尝试着对计算机的结构和指令系统进行改进。20 世纪 70 年代,美国加州伯克利分校开始了对 CISC 指令系统合理性问题的研究,归纳出 CISC 指令系统存在以下 3 个方面的问题。

(1) "8020 规律":即在 CISC 指令系统的计算机中,20%的指令在各种应用程序中的出现频率占整个指令系统的 80%。

(2) CISC 指令系统中有大量的复杂指令,控制逻辑极不规整,给 VLSI 工艺造成很大的困难。

(3) CISC 中增加了许多复杂指令,这些指令虽然简化了目标程序、缩小了高级语言与机器语言之间的差距,但使程序总的执行时间变长、硬件的复杂度增加。

基于这些研究,人们提出了精简指令系统计算机(RISC)。RISC 目前还是一种计算机体系结构的设计思想,不是一种产品,它是近代计算机体系结构发展史中的一个里程

碑。它的核心思想是通过简化指令来使计算机的结构更加简单、合理,从而提高 CPU 的运算速度。卡内基·梅隆(Carnegie Mellon)大学对 RISC 的特点给出了一个较为明确的描述。

(1) 大多数指令在一个计算机周期内完成。所谓计算机周期,是指由寄存器取两个操作数并完成一次算术逻辑运算操作,然后再将运算结果写入寄存器所需的时间。

(2) 因为访问存储器指令需要的时间比较长,因此指令系统中应尽量减少这类指令,而采用寄存器与寄存器之间的操作。

(3) 减少寻址方式的种类。在一个 RISC 内,几乎所有的指令都使用寄存器寻址方式。其他的更为复杂的寻址方式可以通过软件的方法用这些简单的寻址方式予以合成来解决。

(4) 减少指令的种类。指令系统中的大多数指令只执行一个简单的和基本的功能。对复杂的功能,可通过软件编程的方法解决。

(5) 指令格式简单。通常 RISC 仅配备有一种或少数几种指令格式,且指令长度是固定的,并与字节的边界对准;字段位置,特别是操作码字段的位置是固定的。这样处理的好处是:对固定字段、对操作码的译码和对寄存器操作数的访问可同时进行;简化了指令的格式,也就简化了控制器;同时,以字长的单位来取指令和数据,取指令操作过程也就被优化了。

总之,RISC 的特点是简化了计算机的指令系统,进而简化了控制器。如一个 RISC 指令系统可以只有一条或两条 ADD 指令(仅有整数加、带进位加),而 CISC 结构的 Pentium 微处理机仅加法指令就有 4 条。

虽然 RISC 指令功能简单,复杂功能需要用软件编程去实现,但经过技术测试比较,处于同样工艺水平的芯片,RISC 的运算速度要比 CISC 快 3~5 倍。

设计 RISC 类计算机的目的是提高整个系统的性能。要达到这个目的,必须要有相应的技术支持。

① 要求大多数操作使用寄存器操作数,从根本上提高 CPU 的运算速度;

② 指令采用流水线工作方式,取指令和执行指令并行执行,并通过相应的技术手段使流水线尽量不“断流”。具体地说,就是一条指令的执行是由若干个不同的功能子部件分别完成的,流水线中的若干个功能子部件按照指令的执行步骤各自完成自己的操作。如果在程序执行过程中遇到后一条指令要用到前一条指令的执行结果或程序转移情况等,可通过编译程序予以解决,即在编译程序对用高级语言编写的应用程序进行编译时,事先把机器指令的执行顺序安排好,以便最大限度地挖掘流水线的能力。

RISC 类微处理器对存储器的结构和存取速度要求很高,所以在 RISC 系统中一定要采用 Cache,以便减少争用 RISC 芯片的要求。

3.2 寻址方式

所谓寻址方式,主要是指获得操作数所在的地址的方法。在 8088/8086 系统中,一般将寻址方式分为两种不同的类型:

① 寻找操作数的地址；

② 寻找要执行的下一条指令的地址，即程序的地址。后者主要在程序转移或过程调用时用来寻找目标地址或入口地址，这将在调用指令(CALL)和程序转移指令(JMP)中介绍。在3.2节中，主要讨论针对操作数地址的寻址方式，并且如无特殊声明，讨论的对象主要是源操作数。

在8086指令系统中，说明操作数所在地址的寻址方式可分为8种，了解什么样的寻址方式适用于什么样的指令，对于正确理解和合理使用指令是很重要的。

3.2.1 立即寻址

立即寻址(Immediate Addressing)方式只针对源操作数。此时源操作数是一个立即数，它作为指令的一部分，紧跟在指令的操作码之后、存放于内存的代码段中，在CPU取指令时随指令码一起取出并直接参加运算。这里的立即数可以是8位或16位的整数。若为16位，则存放时低8位在低地址单元存放，高8位在高地址单元存放，如图3-2所示。

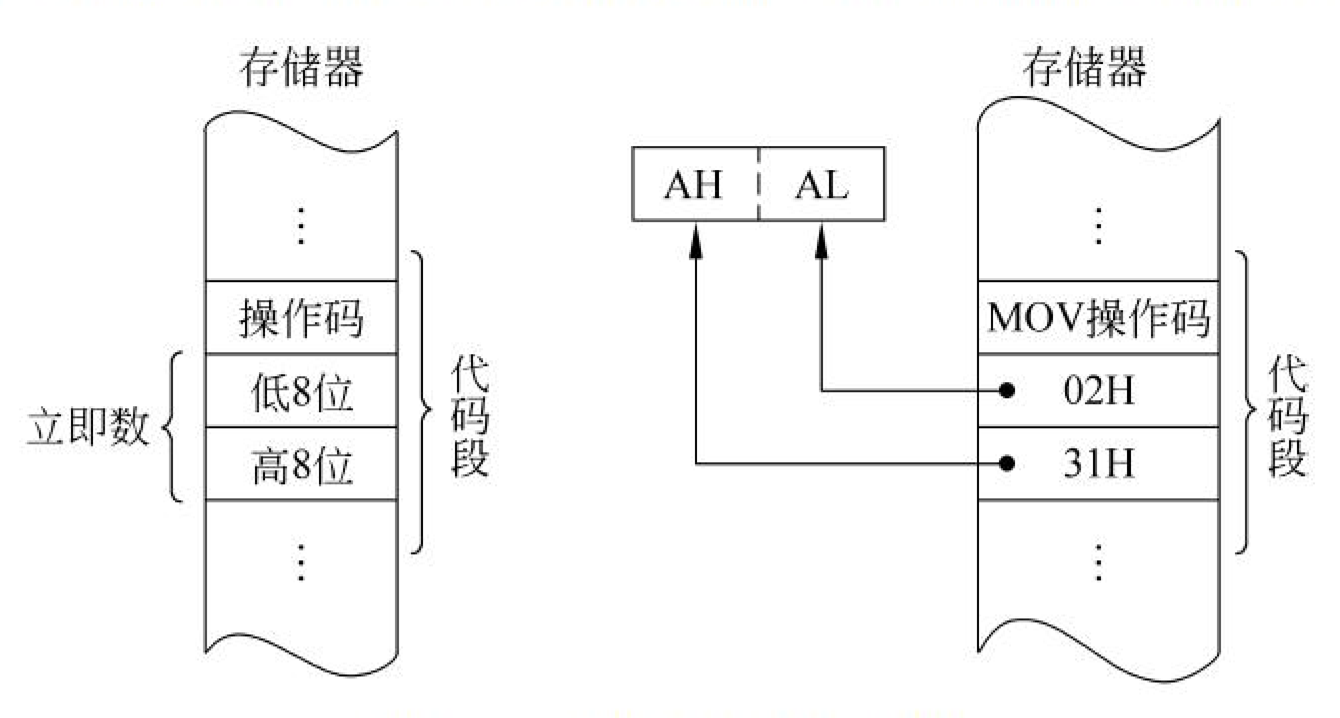


图3-2 立即寻址方式示意图

【例3-1】 指令“MOV AX,3102H”表示将16位的立即数3102H送入累加器AX。指令执行后，AH=31H，AL=02H。

这是一条3字节指令，其执行情况示意图如图3-2所示。立即寻址方式主要用于给寄存器或存储单元赋初值。

3.2.2 直接寻址

直接寻址(Direct Addressing)方式表示参加运算的数据存放在内存中，存放的地址由指令直接给出，即指令中的操作数是存储器操作数。“[]”内用16位常数表示存放数据的偏移地址，数据的段基地址默认为数据段，可以允许段重设。

【例3-2】 指令“MOV AX,[3102H]”表示将数据段中偏移地址为3102H和3103H两单元的内容送到AX中。

假设DS=2000H，则所寻找的操作数的物理地址为

20000H+3102H=23102H

指令的执行情况如图 3-3 所示。

要注意区别直接寻址指令与前面介绍的立即寻址指令二者的不同。直接寻址指令中的数值是操作数的 16 位偏移地址,而不是数据本身。为了区分二者,指令系统规定偏移地址必须用方括号括起来。如在例 3-2 中,指令的执行不是将立即数 3102H 送到累加器 AX,而是将偏移地址为 3102H 的内存单元中的内容送到 AX。若操作数不是存放在 DS 段,则在指令中要用段重设符号加以声明。

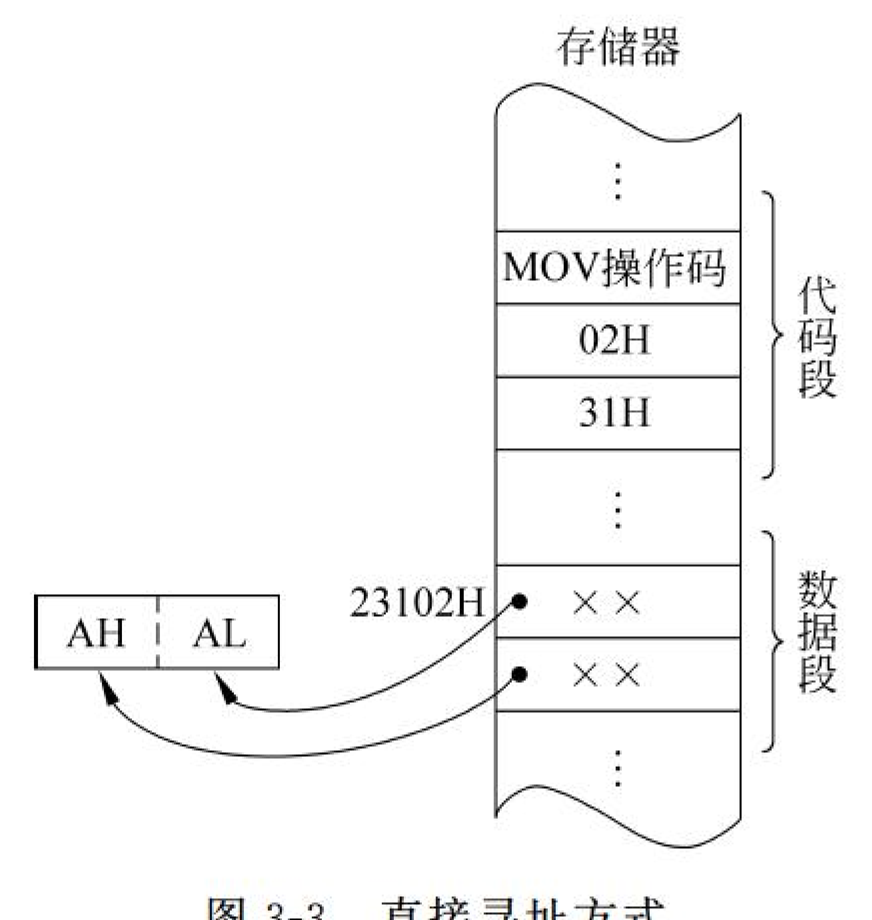


图 3-3　直接寻址方式

【例 3-3】 指令“MOV BL,ES:[1200H]”表示将附加段中偏移地址为 1200H 单元的内容送到 BL 寄存器中。

在汇编语言中,有时也用一个符号来代替数值以表示操作数的偏移地址,通常把这个符号称为符号地址。例 3-3 中,若用 BUFFER 代替偏移地址 1200H,则指令可写成

```
MOV BL,ES:[BUFFER]
```

这两者是等效的,但 BUFFER 必须在程序的开始处予以定义,这点将在第 4 章中介绍。

3.2.3 寄存器寻址

在寄存器寻址(Register Addressing)方式下,指令的操作数为 CPU 的内部寄存器。它们可以是数据寄存器(8 位或 16 位),也可以是地址指针、变址寄存器或段寄存器。

【例 3-4】 指令“MOV SI,AX”表示将 AX 的内容送到寄存器 SI 中。若指令执行前 AX=2233H,SI=4455H,则指令执行后 SI=2233H,而 AX 中的内容保持不变,如图 3-4 所示。

SI　2233H ← AX　2233H

图 3-4　寄存器寻址示意图

采用寄存器寻址方式,虽然指令操作码在代码段中,但操作数在内部寄存器中,指令执行时不必通过访问内存就可取得操作数,故执行速度较快。

3.2.4 寄存器间接寻址

寄存器间接寻址(Register Indirect Addressing)是用寄存器的内容表示操作数的偏移地址。此时寄存器中的内容不再是操作数本身,而是存放数据的偏移地址,操作数本身在内存储器中。

寄存器间接寻址方式中存放操作数偏移地址的寄存器只允许是 SI、DI、BX 和 BP,它

们可简称为间址寄存器或称为地址指针。选择不同的间址寄存器涉及的段寄存器不同。在默认情况下，选择 SI、DI、BX 作间址寄存器时，操作数在数据段，段基地址由 DS 决定；选择 BP 作间址寄存器，则操作数在堆栈段，段基地址由 SS 决定。但无论选择哪一个间址寄存器都允许段重设，可在指令中用段重设符指明当前操作数在哪一个段。

因为间址寄存器中存放的是操作数的偏移地址，所以指令中的间址寄存器必须加上方括号，以避免与寄存器寻址指令混淆。

【例 3-5】 已知 DS＝6000H，SI＝1200H，执行指令：MOV AX,[SI]。

因为指令中没有指定段重设，所以寻址时使用默认的段寄存器 DS。由已知条件可计算出操作数的物理地址＝60000H＋1200H＝61200H。指令执行情况如图 3-5 所示。

执行结果：AX＝3344H。

若操作数存放在附加段，则本例中的指令应表示成以下形式：

```
MOV AX,ES:[SI]
```

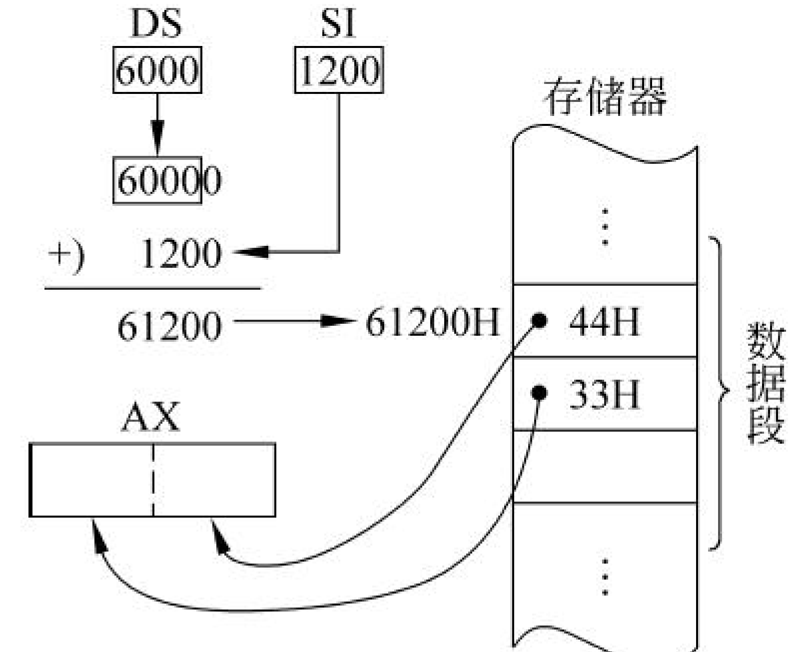


图 3-5 寄存器间接寻址示意图

例 3-5 中，若间址寄存器采用 BP，则操作数默认存放在堆栈段。

【例 3-6】 若已知 SS＝8000H，BP＝0200H，指令“MOV BX,[BP]”执行后：BL＝[80200H]单元中的内容，BH＝[80201H] 单元中的内容。

有些书中又将使用 BX、BP 作为间址寄存器的寄存器寻址方式称为基址寻址方式；而将使用 SI、DI 作为间址寄存器的寄存器寻址方式称为变址寻址方式。

3.2.5 寄存器相对寻址

在寄存器相对寻址方式中，操作数在内存中的存放地址(偏移地址)由间址寄存器的内容加上指令中给出的一个 8 位或 16 位的位移量组成。操作数所在段由所使用的间址寄存器决定(规则与寄存器间接寻址方式相同)。因位移量可看作相对值，故把这种带位移量的寄存器间接寻址方式称为寄存器相对寻址。

【例 3-7】 指令 MOV AX,DATA[BX]的寻址过程示例。

设：DS＝6000H，BX＝1000H，DATA＝0008H。

则操作数所在单元的物理地址＝60000H＋1000H＋0008H＝61008H。

执行结果：AX＝5566H。

指令的执行情况如图 3-6 所示。

寄存器相对寻址常用于存取表格或一维数组中的元素——把表格的起始地址作为位移量，元素的下标值放在间址寄存器中(反过来也可以)。这样，就可存取表格中的任意一个元素。

【例 3-8】 某数据表的首地址(偏移地址)为 TABLE，要取出该表中的第 10 个字节并存放到 AL 中，可用如下指令段实现(注意位移量是从 0 开始的)：

```
MOV SI,9                    ;第 10个数的位移量为 9
MOV AL,[TABLE+ SI]          ;第 10个数的偏移地址为 TABLE+ 9
```

在汇编语言中,相对寻址指令的书写格式允许有几种不同的形式。例如,以下几种写法实质上是完全等价的。

```
MOV AL,DATA[SI]
MOV AL,[SI]DATA
MOV AL,DATA+ [SI]
MOV AL,[SI]+ DATA
MOV AL,[DATA+ SI]
MOV AL,[SI+ DATA]
```

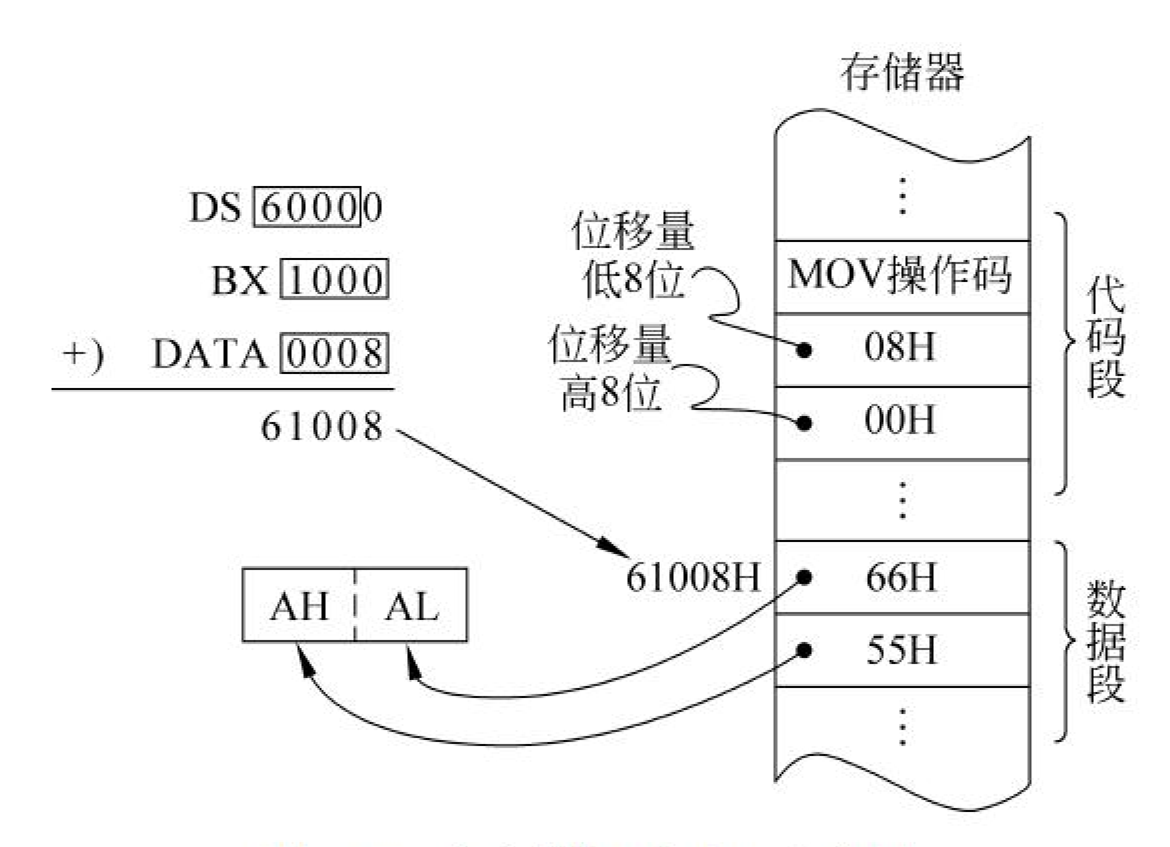


图 3-6　寄存器相对寻址示意图

3.2.6　基址—变址寻址

基址—变址寻址方式由一个基址寄存器(BX 或 BP)的内容和一个变址寄存器(SI 或 DI)的内容相加而形成操作数的偏移地址,称为基址—变址寻址。在默认的情况下,指令中若用 BX 作基址寄存器,则段地址在 DS 中;如果用 BP 作基址寄存器,则段地址在 SS 中,但允许使用段重设。

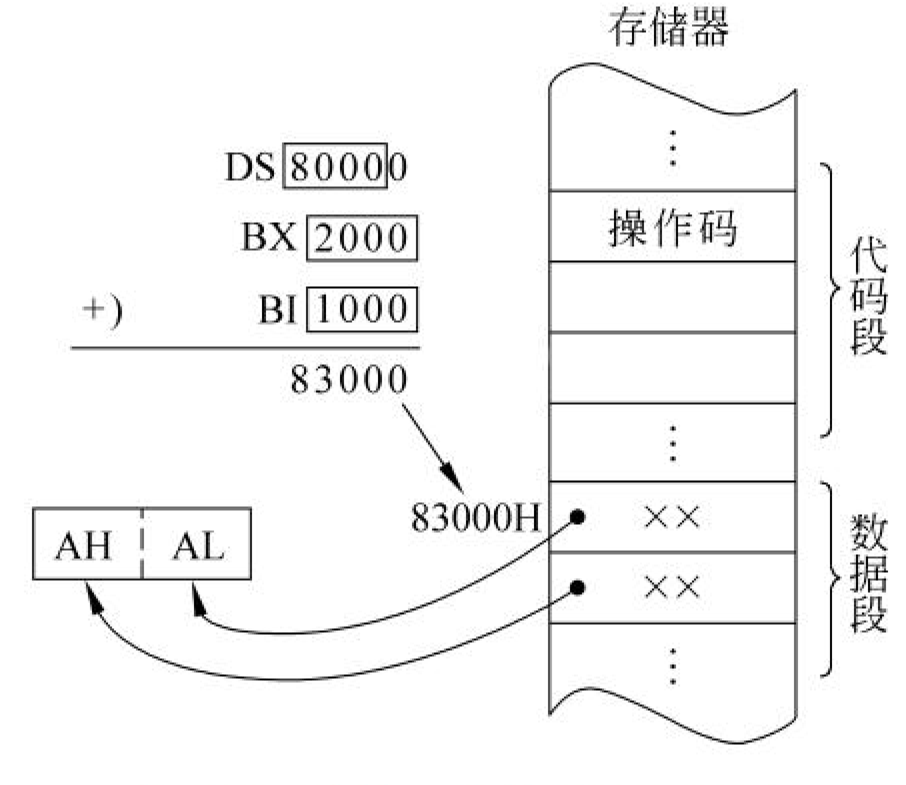


图 3-7　基址—变址寻址示意图

【例 3-9】　指令 MOV AX,[BX][SI]的寻址过程如图 3-7 所示。

设:DS=8000H,BX=2000H,SI=1000H。

则操作数的物理地址=80000H+2000H+1000H=83000H。

指令执行后:AL=[83000H],AH=[83001H]。

注意：使用基址—变址方式时，不允许将两个基址寄存器或两个变址寄存器组合在一起寻址，即指令中不允许同时出现两个基址寄存器或两个变址寄存器。例如，以下指令是非法的。

```
MOV AX,[BX][BP]          ;错误!同时出现两个基址寄存器
MOV AX,[SI][DI]          ;错误!同时出现两个变址寄存器
```

3.2.7 基址—变址—相对寻址

基址—变址—相对寻址方式事实上是基址—变址寻址方式的扩充。指令中指定一个基址寄存器和一个变址寄存器，同时还给出一个8位或16位的位移量，将三者相加就得到操作数的偏移地址。至于默认的段寄存器仍由所用的基址寄存器决定，指令允许使用段重设。

【例 3-10】 指令 MOV AX,5[DI][BX]的寻址过程示例。

该指令将段地址为DS、偏移地址为BX＋DI＋5的连续两个存储单元的内容送到AX。指令执行情况的示意图如图3-8所示。

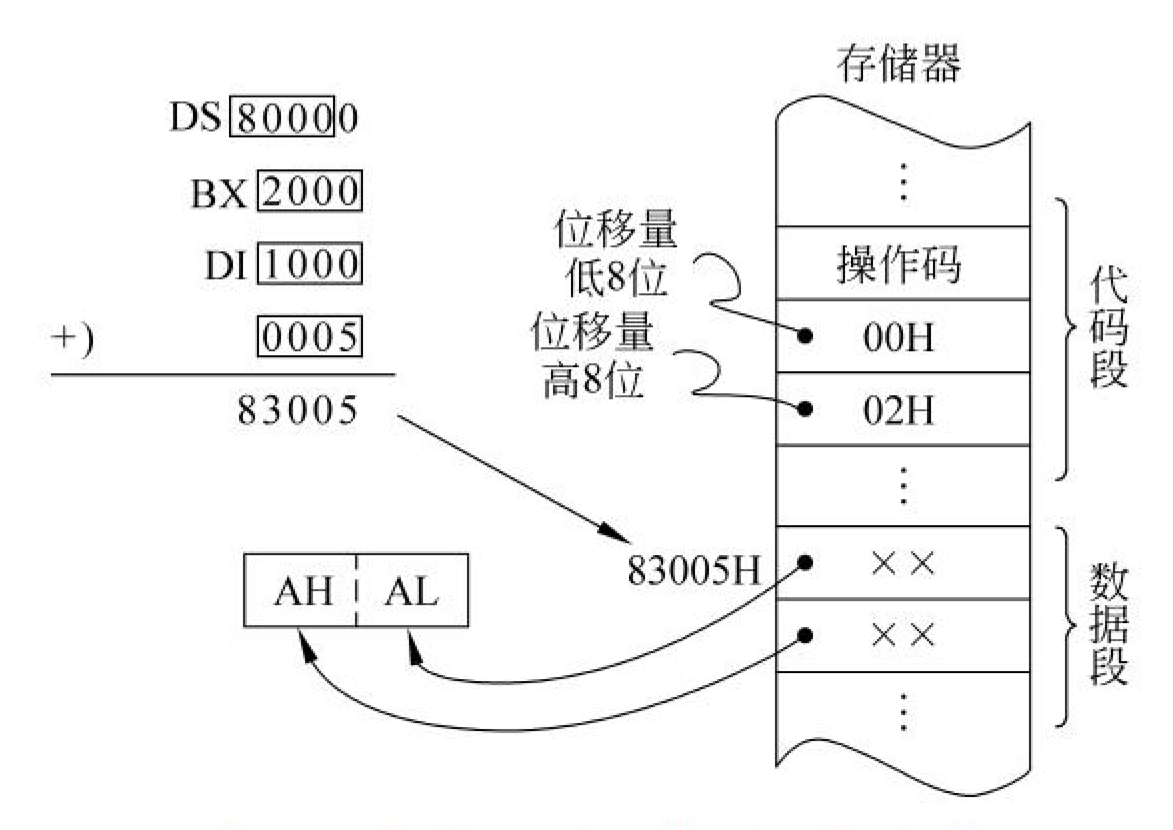


图 3-8 基址—变址—相对寻址示意图

使用这种寻址方式可以很方便地访问二维数组。例如，用基址寄存器存放数组的首地址（偏移地址），而变址寄存器和位移量分别存放行和列的值，指令就可以直接访问二维数组中指定的行和列的元素。

与寄存器间接寻址方式类似，基址—变址—相对寻址指令同样也可以表示成多种形式，例如：

```
MOV AX,DATA[SI][BX]
MOV AX,[BX+DATA][SI]
MOV AX,[BX+SI+DATA]
MOV AX,[BX]DATA[SI]
MOV AX,[BX+SI]DATA
```

同样地，基址—变址—相对寻址也不允许在指令中同时出现两个基址寄存器或两个变址寄存器，即下列指令也是非法的。

```
MOV AX,DATA[SI][DI]          ;错误!同时出现两个变址寄存器
MOV AX,[BX][BP]DATA          ;错误!同时出现两个基址寄存器
```

3.2.8 隐含寻址

有些指令的操作码中不仅包含了操作的性质，还隐含了部分操作数的地址。如乘法指令 MUL，在这条指令中只需指明乘数的地址，而被乘数以及乘积的地址是隐含且固定的。这种将一个操作数隐含在指令码中的寻址方式就称为隐含寻址。

【例 3-11】 指令 MUL BL 的功能是把 AL 中的内容与 BL 中的内容相乘，乘积送到 AX 寄存器，即 AL×BL→AX。这条指令隐含了被乘数 AL 及乘积 AX。

3.3 8086 指令系统

本节将详细介绍 8086 指令系统。这里首先给出以下介绍中要用到的一些符号。

OPRD	泛指各种类型的操作数
mem	存储器操作数
acc	累加器操作数
dest	目标操作数
src	源操作数
disp	8位或 16位偏移量，可用符号地址表示
DATA	8位或 16位立即数
port	输入输出端口，可用数字或表达式表示
[]	表示存储器操作数，方括号中的内容表示数据的偏移地址

3.3.1 数据传送指令

数据传送指令是实际程序中使用最为频繁的一类指令，因为无论什么样的程序都需要将原始数据、中间运算结果、最终结果及其他信息在 CPU 的寄存器和存储器之间进行传送。绝大多数数据传送指令都不会对状态寄存器 FLAGS 产生影响。

数据传送类指令按功能可分为四小类：通用数据传送指令、目标地址传送指令、标志传送指令、输入输出指令。

1. 通用数据传送指令

通用数据传送指令包括一般传送指令 MOV、堆栈操作指令 PUSH 和 POP、交换指令 XCHG、查表转换指令 XLAT 和字位扩展指令。

1）一般传送指令 MOV

指令格式及操作：

```
MOV dest,src                    ;(dest)← (src)
```

这里，dest 表示目标操作数，src 表示源操作数。指令的功能是将一个操作数从源地址传送到目标地址，而源地址中的数据保持不变。也就是说，MOV 指令实际上是完成了一次数据的复制。

在汇编语言中，规定具有双操作数的指令必须将目标操作数写在前面，源操作数写在后面，两者之间用一个逗号隔开。

（1）指令特点。MOV 指令是最普通、最常用的传送指令，它具有如下几个特点。

① 指令中的操作数可以是 8 位，也可以是 16 位。一次传送的数据到底是字节还是字取决于指令中涉及的寄存器是 8 位还是 16 位的。

② 可以使用 3.2 节讨论过的各种寻址方式。

（2）指令实现的操作。MOV 指令可以实现以下各种传送。

① 寄存器与寄存器或寄存器与段寄存器之间的传送。例如：

```
MOV BX,SI                       ;将变址寄存器 SI 中的内容送到基址寄存器 BX
MOV DS,AX                       ;将累加器 AX 中的内容送到段寄存器 DS
MOV AL,CL                       ;将通用寄存器 CL 中的内容送 AL
```

② 寄存器与存储器之间的传送。MOV 指令可以在寄存器与存储器之间进行数据传送。若传送的是字操作数，那么将对连续两个存储器单元进行存取，且寄存器的高 8 位对应存储器的高地址单元，寄存器的低 8 位对应存储器的低地址单元。例如：

若有 DS＝6000H，SS＝8000H，AX＝1234H，BX＝1200H，DI＝0383H，BP＝1020H，则有

```
MOV [BX],AX                  ;将 AX 的内容送内存单元。其中 [61200H]= 34H,[61201H]= 12H
MOV CL,[BP][DI]              ;将堆栈段中偏移地址为 BP+ DI= 13A3H 单元的内容送 CL
                             ;即物理地址为 813A3H 单元的内容送 CL
MOV AX,[6000H]               ;将 DS 段的 6000H 和 6001H 两个单元的内容送 AX
```

③ 立即数到寄存器的传送。

```
MOV AL,5                        ;将立即数 5 送累加器 AL
MOV BX,3078H                    ;将立即数 3078H 送寄存器 BX
```

④ 立即数到存储器的传送。

```
MOV BYTE PTR[BP+ SI],5          ;将 5 送堆栈段中偏移地址为 BP+ SI 所指的单元中
MOV WORD PTR[BX],1005H          ;将 1005H 送数据段中偏移地址为 BX 和 BX+ 1 两单元
```

⑤ 存储器与段寄存器之间的传送。

```
MOV DS,[1000H]                  ;将数据段中偏移地址为 1000H 字单元内容送数据段寄存器 DS
MOV [BX],ES                     ;将附加段寄存器 ES 内容送数据段中 BX 所指向的字单元
```

(3) 指令对操作数的要求。

① MOV 指令中两个操作数字长必须相同。两个操作数可同为字节数或同为字操作数。

② 两个操作数不能同时为存储器操作数。若要在两个存储器单元之间进行数据传送,需用两条 MOV 指令实现。

③ 不能用立即数直接给段寄存器赋值。要实现此功能,需使用两条 MOV 指令。

④ 两个操作数不能同时为段寄存器。同样,要实现段寄存器到段寄存器的数据传送,需用两条 MOV 指令。

⑤ 一般情况下,指令指针 IP 及代码段寄存器 CS 的内容不通过 MOV 指令修改,即它们不能作为目标操作数,但可以作为源操作数。

⑥ 虽然许多指令的执行都对状态寄存器 FLAGS 的标志位产生影响,但通常情况下,FLAGS 整体不能作为操作数。

实际编写程序中,有时需要将内存一个区域中若干单元的数据(称为数据块)传送到另外一个区域或是向若干单元赋同样的值(如清零)。对于这种重复性的工作,计算机是最乐意做的。下面就通过一个例子来说明如何利用 MOV 指令完成数据块的传送。

【例 3-12】 把内存中首地址为 MEM1 的 200 个字节送到首地址为 MEM2 的区域中。

题目分析:

两个内存单元间的数据传送需要用两条 MOV 指令实现,在这里当然不希望用 400 条 MOV 指令来完成这 200 个单元数据的传送。较好的实现方式是通过循环程序来实现这个数据块的传送。下面的程序段中某些指令还没有学到,这里先拿来用用。

```
        MOV SI,OFFSET MEM1      ;源数据块首地址(偏移地址)送 SI
        MOV DI,OFFSET MEM2      ;目标首地址(偏移地址)送 DI
        MOV CX,200              ;数据块长度送 CX,即=循环次数 CX
NEXT:   MOV AL,[SI]             ;源数据块中当前字节送 AL
        MOV [DI],AL             ;AL 内容送目标地址,完成一个字节数据的传送
        INC SI                  ;SI 加 1,修改源地址指针
        INC DI                  ;DI 加 1,修改目标地址指针
        DEC CX                  ;CX 减 1,修改循环次数
        JNZ NEXT                ;若循环次数(CX)不为零,则转移到 NEXT 标号处
        HLT                     ;停止
```

2) 堆栈操作指令 PUSH 和 POP

(1) 堆栈的概念。堆栈是内存中一个特定的区域,用以存放寄存器或存储器中暂时不用又必须保存的数据。它在内存中所处的段称为堆栈段,其段地址放在堆栈段寄存器 SS 中。可以将堆栈看作是一个小存储器,但不能任意存取,必须遵循以下的原则。

① 堆栈的存取每次必须是一个字(16 位),即堆栈指令中的操作数必须是 16 位,而且只能是寄存器或存储器操作数,不能是立即数。

② 向堆栈中存放数据时,总是从高地址向低地址方向增长,而从堆栈取数据时则方

向正好相反。

③ 堆栈段在内存中的位置由 SS 决定，堆栈指针 SP 总是指向栈顶，即 SP 的内容等于当前栈顶的偏移地址。所谓栈顶是指当前可用堆栈操作指令进行数据交换的存储单元，如图 3-9 所示。在压入操作数之前，SP 先减 2，每弹出一个字，SP 加 2。

④ 对堆栈的操作遵循“后进先出（LIFO）”的原则。

在程序中，堆栈主要应用于子程序调用、中断响应等操作时的参数保护，也可用于实现参数传递。

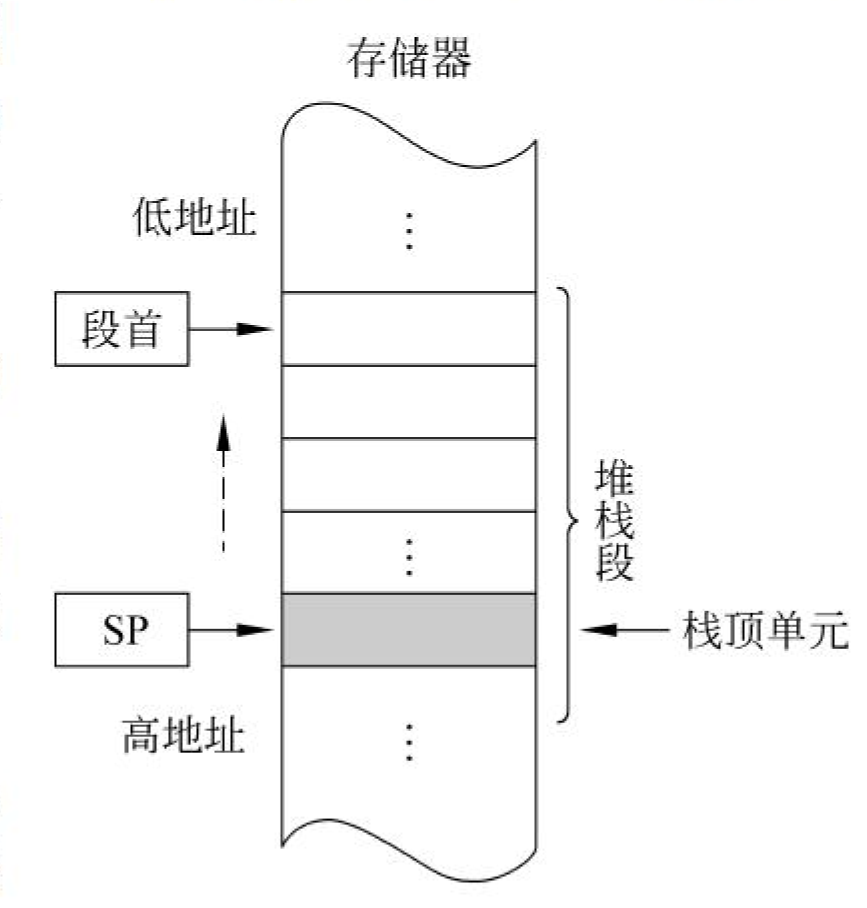


图 3-9　堆栈区示意图

（2）堆栈操作指令。堆栈操作指令共有两条：压入堆栈（压栈）指令 PUSH 和弹出堆栈（出栈）指令 POP。其格式为

```
PUSH   src
POP    dest
```

指令中的操作数 src 和 dest 必须为字操作数（16 位），它们可以是：①16 位的通用寄存器或段寄存器（CS 除外，PUSH CS 指令是合法的，而 POP CS 指令是非法的）；②存储器单元（地址连续的两个存储单元）。

例如：

```
PUSH AX                          ;通用寄存器内容压入堆栈
PUSH WORD PTR[DATA+ SI]          ;数据段中两个连续存储单元内容压入堆栈
POP DS                           ;从栈顶弹出一个字到段寄存器
POP WORD PTR[BX]                 ;从栈顶弹出一个字到数据段两个连续存储单元中
```

（3）堆栈指令的执行过程。

① 压栈指令 PUSH OPRD。PUSH 指令是将指令中指定的字操作数压入堆栈。指令的执行过程为

```
SP- 2→SP
OPRD高 8位→[SP+ 1];
OPRD低 8位→[SP];
```

图 3-10 表示了执行 PUSH AX 指令前后堆栈区的变化情况。这里假设 AX＝1122H。由图 3-10 可见，PUSH 指令是将 16 位的源操作数送到堆栈的顶部。

② 出栈指令 POP OPRD。POP 指令是将当前栈顶的一个字送到指定的目标地址，并紧接着修改堆栈指针，以使 SP 指向新的栈顶位置。指令的执行过程为

```
[SP]→OPRD低 8位
[SP+ 1]→OPRD高 8位
SP+ 2→SP
```

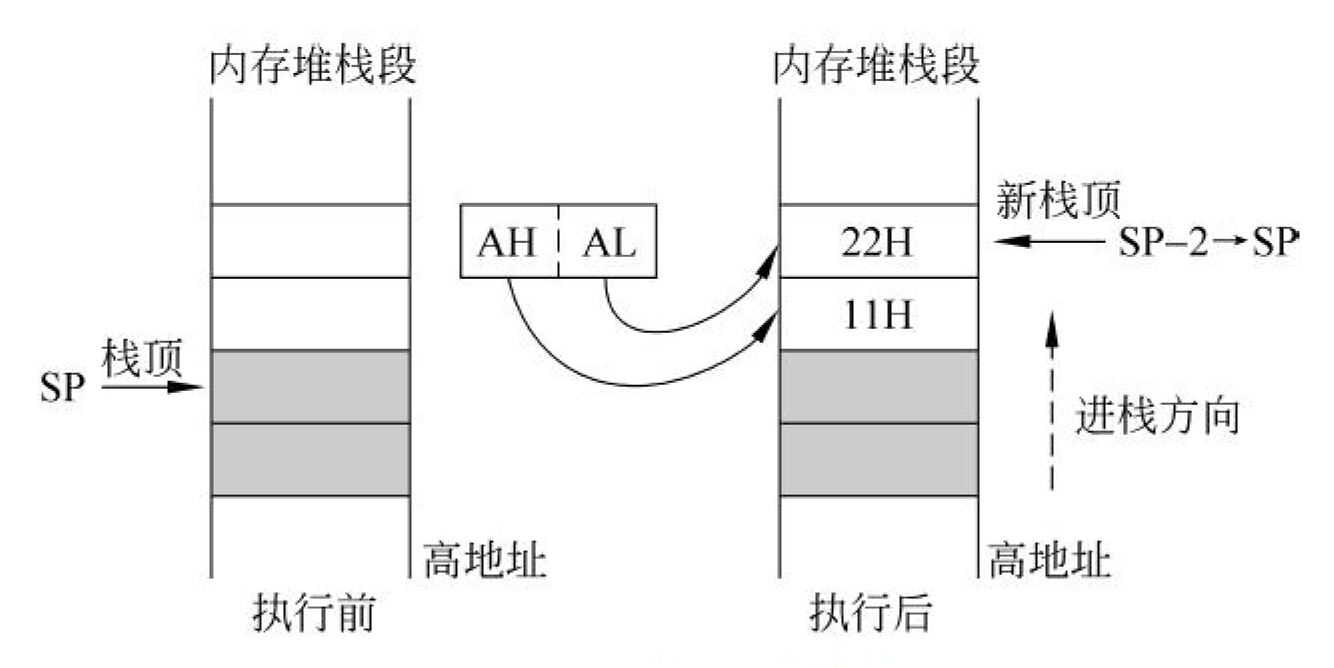


图 3-10　PUSH AX 指令执行示意图

图 3-11 给出了执行 POP AX 指令前后堆栈区的变化情况。这里依然设 AX=1122H。

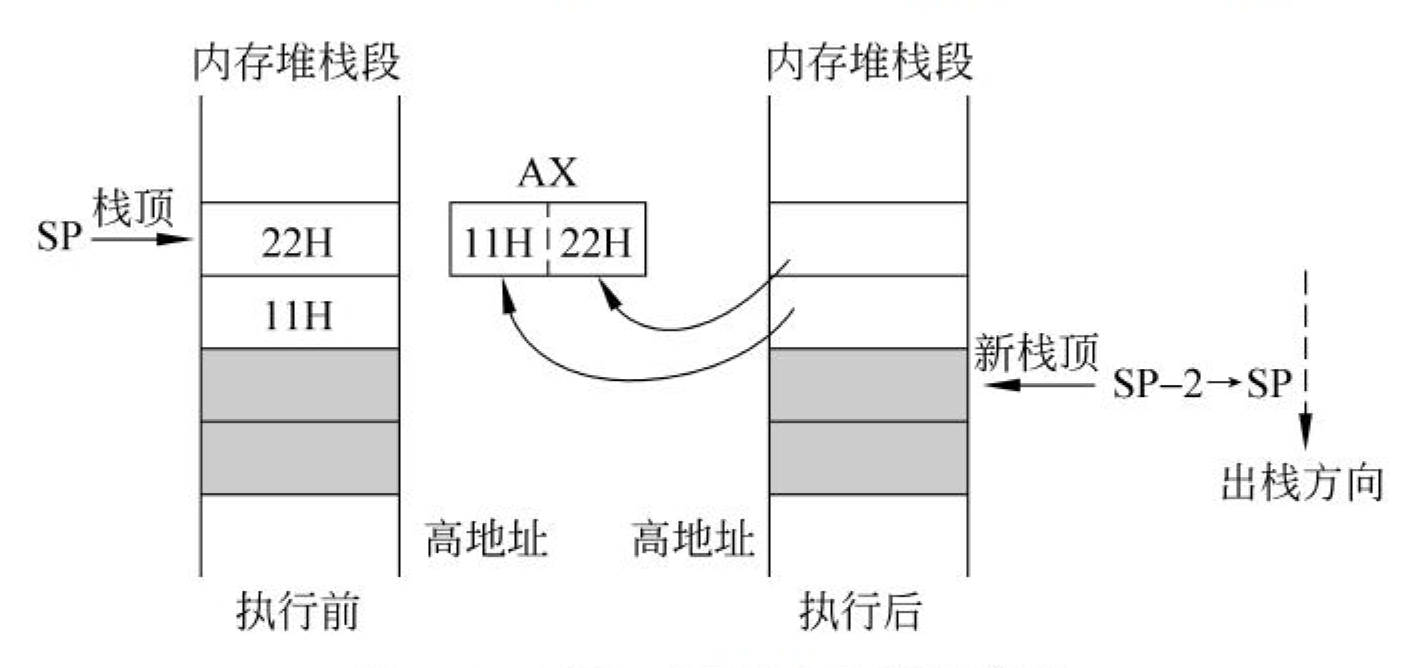


图 3-11　POP AX 指令执行示意图

在程序中，PUSH 和 POP 指令一般成对出现，且执行顺序相反，以保持堆栈原有状态。当然，在必要时也可通过修改 SP 的值来恢复堆栈原有状态。

【例 3-13】　按“先进先出”原则进行堆栈操作的程序例。其执行示意图如图 3-12 所示。

```
MOV AX,9000H
MOV SS,AX
MOV SP,0E200H
MOV DX,38FFH
PUSH DX
PUSH AX
⋮
POP DX
POP AX
```

例 3-13 中 PUSH 和 POP 指令的执行顺序未遵循“后进先出”原则，结果出栈后 AX 和 DX 的内容就没有保持压栈前的状态，而是进行了互换(有时可利用堆栈的这一特点实现两操作数内容互换)。

堆栈除在子程序调用和响应中断时用于保护断点地址外，还可在需要时对某些寄存器内容进行保存。例如，用 CX 寄存器同时作为两重循环嵌套的计数器，可先将外循环计

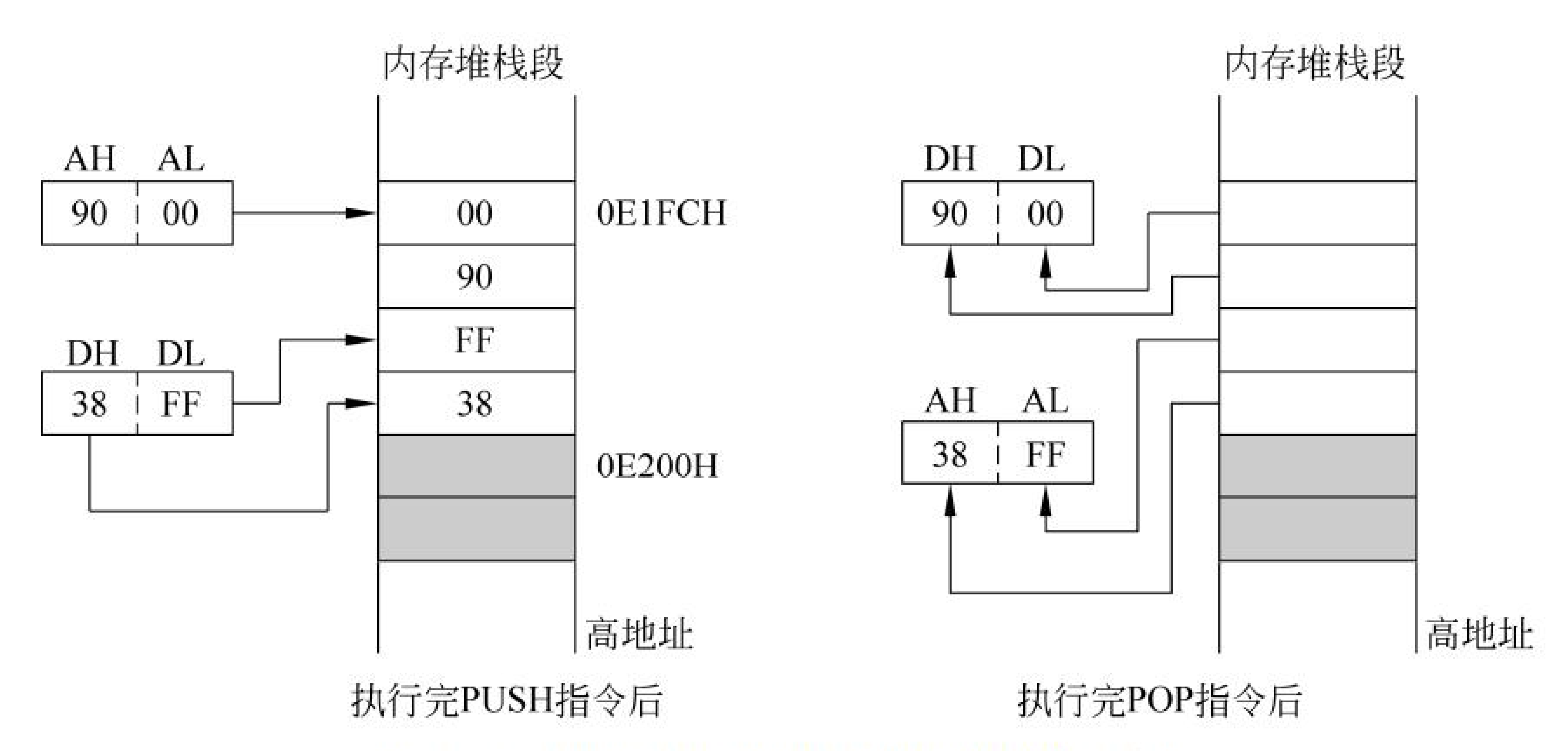


图 3-12 按“先进先出”原则的堆栈操作示意图

数值送 CX,当内循环开始时将 CX 中的外循环计数值压入堆栈保存,然后把内循环计数值写入 CX,内循环完成后再将外循环计数值从堆栈中弹出到 CX。

3)交换指令 XCHG

指令格式及操作:

```
XCHG   OPRD1,OPRD2                ;(OPRD1)←→(OPRD2)
```

交换指令的操作是将源地址与目标地址中的内容进行互换,即将源操作数送到目标操作数,同时将目标操作数传送到源操作数。

交换指令对操作数有如下要求。

(1)源操作数和目标操作数可以是寄存器或存储器,但不能同时为存储器。

(2)不能为段寄存器操作数,即段寄存器的内容不能参加交换。

(3)两个操作数字长必须相同,可以是字节交换,也可以是字交换。例如:

```
XCHG AX,BX                 ;AX→BX,BX→AX
XCHG CL,DL                 ;CL→DL,DL→CL
```

【例 3-14】 设 DS=2000H,SI=0230H,DL=88H,[20230H]=44H,执行指令:

```
XCHG [SI],DL
```

执行结果为[20230H]=88H,DL=44H。DL 的内容与[20230H]的内容进行了交换。

4)查表转换指令 XLAT

XLAT 是一条字节的查表转换指令,可以根据表中元素的序号查出表中相应元素的内容。

预先将要查找的代码排成一个表放在内存某区域中。指令要求将表的首地址(偏移地址)送寄存器 BX,要查找的元素的序号送 AL(表中第一个元素的序号为 0,然后依次为 1、2、3、…)。执行 XLAT 指令后,表中指定序号的元素被存入 AL。

指令格式为

```
XLAT                    ;将偏移地址为 BX+AL所指单元的内容送到 AL中
```

或：

```
XLAT  src_table         ;(src_table表示要查找的表的首地址)
```

利用 XLAT 指令实现查表转换的操作十分方便。

【例 3-15】 在内存的数据段中存放有一张数值为 0～9 的 ASCII 码转换表，首地址为 Hex_table，如图 3-13 所示。现要把数值 8 转换成对应的 ASCII 码，可用以下几条指令实现。

```
LEA BX,Hex_table        ;BX←表首偏移地址
MOV AL,8                ;AL←8
XLAT                    ;查表转换
```

结果 AL＝38H，为 8 所对应的 ASCII 码。

由于要查找元素的序号放在 AL 中，所以表格的最大长度不能超过 256 个字节。

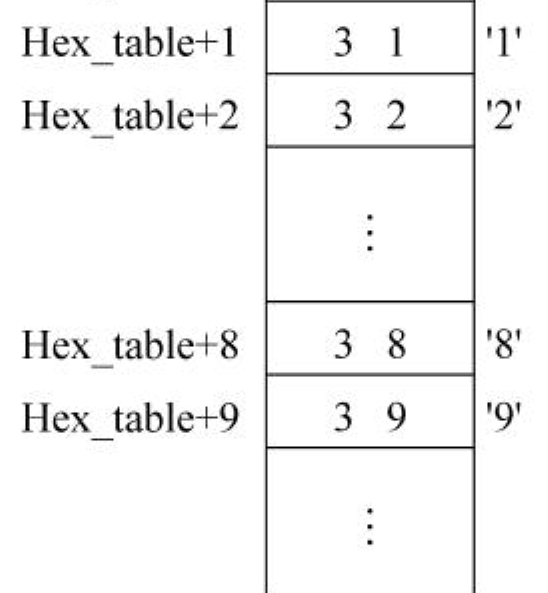


图 3-13　0～9 的换码表

2. 输入输出指令

输入输出(I/O)指令是专门面向输入输出端口进行读写的指令，共有两条：IN 和 OUT。输入指令 IN 用于从 I/O 端口读数据到累加器 AL(或 AX)中，而输出指令 OUT 用于把累加器 AL(或 AX)的内容写到 I/O 端口，即从 CPU 方面看，只有累加器 AL(或 AX)才能与 I/O 端口进行数据传送，所以这两条指令也称为累加器专用传送指令。

8088 系统可连接多个外设端口，可以像存储器一样用不同的地址来区分它们。在 8088 的 I/O 指令中，允许用两种形式来表示端口地址，或称为两种寻址方式。

(1) 直接寻址：指令中的 I/O 端口地址为 8 位，此时允许寻址 256 个端口，端口地址范围为 0～FFH。

(2) 寄存器间接寻址：端口地址为 16 位，由 DX 寄存器指定，可寻址 64K 个端口，地址范围为 0～FFFFH。

间接寻址方式的适用范围较大，在编制程序时要尽量采用这种方式。

1) 输入指令 IN

指令格式：

```
IN acc,port             ;直接寻址,port为用 8位立即数表示的端口地址
```

或

```
IN acc,DX               ;间接寻址,16位端口地址由 DX给出
```

IN 指令从端口输入一个字节到 AL 或输入一个字到 AX 中。

【例 3-16】

```
MOV DX,03B0H            ;将 16位端口地址送 DX
```

```
IN AL,DX                              ;从地址为 3B0H 的端口输入一个字节到 AL
IN AX,3FH                             ;从地址为 3FH 的端口输入一个字到 AX
```

2）输出指令 OUT

指令格式：

```
OUT port,acc                          ;直接寻址,port 为 8 位立即数表示的端口地址
```

或

```
OUT DX,acc                            ;间接寻址,16 位端口地址由 DX 给出
```

OUT 指令将 AL(或 AX)的内容输出到指定的端口。

【例 3-17】

```
OUT 43H,AL                            ;将 AL 的内容输出到地址为 43H 的端口
OUT 44H,AX                            ;将 AX 的内容输出到地址为 44H 的端口
MOV DX,33FH                           ;端口地址 33FH 送 DX
OUT DX,AL                             ;将 AL 的内容输出到地址为 33FH 的端口
```

注意：采用间接寻址的 IN/OUT 指令只能用 DX 寄存器作为间址寄存器。

3. 取偏移地址指令

指令格式：

```
LEA reg16,mem
```

LEA 指令将存储器操作数 mem 的 16 位偏移地址送到指定的寄存器。这里,源操作数必须是存储器操作数,目标操作数必须是 16 位通用寄存器。因该寄存器常用来作为地址指针,故在此最好选用 4 个间址寄存器之一。

【例 3-18】

```
LEA BX,BUFFER                         ;将内存单元 BUFFER 的偏移地址送 BX
MOV AL,[BX]                           ;取出 BUFFER 中的第一个数据送 AL
MOV AH,[BX+1]                         ;取出 BUFFER 中的第二个数据送 AH
```

【例 3-19】 若设 BX＝1000H,DS＝6000H,[61050H]＝33H,[61051H]＝44H。比较以下两条指令的执行结果。

```
LEA BX,[BX+50H]
MOV BX,[BX+50H]
```

执行过程如图 3-14 所示。第一条指令执行后 BX＝1050H;第二条指令执行后 BX＝4433H。

4. 其他传送指令

除以上传送类指令外,8086 指令系统中还有一些其他的数据传送指令,它们的格式

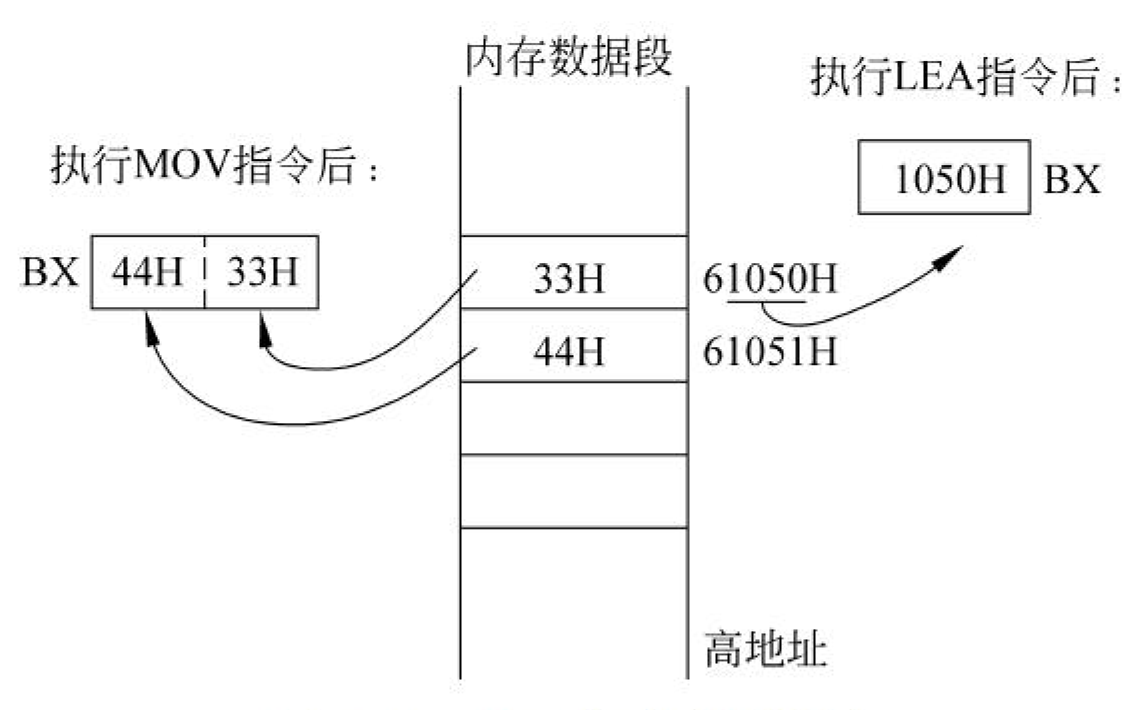


图 3-14 LEA 指令应用举例

和功能如表 3-2 所示。

表 3-2 其他传送类指令

指令类型	汇编格式	指令的操作	示 例
字位扩展指令	CBW	将 AL 中的字节数扩展为字，并存放在 AX 中。扩展的原则是：将符号位扩展到整个高位	MOV AL,8EH CBW ;结果：AX=FF8EH
	CWD	将 AX 中的字扩展为双字，扩展后的高 16 位存放在 DX 中。扩展的原则与 CBW 指令相同	MOV AX,438EH CWD ;结果：AX=438EH,DX=0000H
远地址传送指令	LDS reg16,mem32	mem32 为内存中连续 4 个单元的首地址。指令将[mem32]和[mem32+1]单元的内容送 reg16，将[mem32+2]和[mem32+3]单元的内容送 DS	设 1234H 为首的 4 个单元的内容分别为：11H,22H,00H,90H。则执行完指令： LDS SI,[1234H] ;SI=2211H,DS=9000H
	LES reg16,mem32	指令将[mem32]和[mem32+1]单元的内容送 reg16，将[mem32+2]和[mem32+3]单元的内容送 ES	
标志传送指令	LAHF	将 FLAGS 低 8 位的内容送 AH	设 SF=1,ZF=0,AF=1,PF=1,CF=0 执行指令 LAHF;AH 各位状态为 10×1×1×0 注：×表示任意状态
	SAHF	将 AH 的内容送到 FLAGS 低 8 位	
	PUSHF	将 FLAGS 的内容压入堆栈中保存	
	POPF	将当前栈顶的两个单元的内容弹出到 FLAGS 中	

3.3.2 算术运算指令

8086 提供了加、减、乘、除 4 组基本的算术运算指令，可实现字节或字、无符号数或有符号数的运算。指令对操作数的要求类似于数据传送类指令，即单操作数指令中的操作数不允许使用立即数；在双操作数指令中，立即数只能作为源操作数；不允许源操作数和目的操作数都是存储器等。

算术运算涉及运算结果是否可能溢出。由第 1 章已经知道，无符号数和有符号数的表示方法、数的可表示范围及溢出标志都不一样。有符号数的溢出是一种出错，而无符号数的溢出不能简单地认为是出错，也可看作是向更高位的进位。它们的判断标志分别为 CF 和 OF。

除 4 组二进制的算术运算指令外，8086 还提供了与之对应的 4 类十进制调整指令，可将运算结果调整为以 BCD 码表示的十进制数。

算术运算指令大多会对标志位产生影响，下面分别介绍这 4 组指令。

1. 加法运算指令

加法运算指令有 3 条：普通加法指令 ADD、带进位位的加法指令 ADC 及加 1 指令 INC。其中，双操作数指令对操作数的要求与 MOV 指令基本相同，但有一点：段寄存器不能作为加法指令的操作数。

1）普通加法指令 ADD

指令格式：

```
ADD OPRD1,OPRD2                    ;OPRD1←OPRD1+OPRD2
```

ADD 指令的执行是将源操作数和目标操作数相加，结果送回目标地址中。

这里，源操作数 OPRD2 和目标操作数 OPRD1 均可以是 8 位或 16 位的寄存器或存储器操作数，源操作数还可以是立即数，可以是无符号数，也可以是带符号数。例如：

以下指令是合法的。

```
ADD CL,20H                         ;CL←CL+20H
ADD DX,[BX+SI]                     ;DX←DX+[BX+SI]
```

以下两条指令则是非法的。

```
ADD [SI],[BX]                      ;不允许两个操作数都是存储器操作数
ADD DS,AX                          ;不允许把段寄存器作为操作数
```

ADD 指令的执行对全部 6 个状态标志位都会产生影响。

【例 3-20】

```
MOV AL,7EH                         ;AL←7EH
ADD AL,5BH                         ;AL←7EH+5BH
```

这两条指令执行后，状态标志位的状态分别为

```
AF=1        表示 D3 向 D4 有进位
CF=0        表示最高位向前无进位
OF=1        表示若为有符号数加法,其运算结果产生溢出
PF=0        表示 8位的运算结果中,"1"的个数为奇数
SF=1        表示运算结果的最高位为"1"
ZF=0        表示运算结果不为"0"
```

事实上,指令执行后,AL＝D9H＞7FH(8 位带符号数的最大值),但 D9H＜FFH(8 位无符号数的最大值)。所以有 CF＝0,OF＝1。

2) 带进位位的加法指令 ADC

指令格式:

```
ADC  OPRD1,OPRD2                    ;OPRD1←OPRD1+OPRD2+CF
```

ADC 指令与 ADD 指令在功能、格式及对标志位的影响上都基本相同,只是 CF 也要参加求和运算,结果依然送目标操作数。

【例 3-21】 设 CF＝1,写出以下指令执行后的结果。

```
MOV AL,7EH
ADC AL,0ABH
```

指令执行后:AL＝7EH＋0ABH＋1＝2AH,且 CF＝1。

ADC 指令主要用于多字节加法运算。由于 8086 一次最多只能实现两个 16 位数相加,故对多于两个字节的数的加法,只能先加低 16 位(或低 8 位),再加高 16 位(或高 8 位),但在高位相加时,必须要考虑低位向上的进位,这时就需使用 ADC 指令。

【例 3-22】 求两个 4 字节无符号数 0107A379H 和 10067E4FH 的和。

```
MOV  AX,0A379H        ;第一个数低 16位送 AX
ADD  AX,7E4FH         ;两个数的低 16位相加,结果送 AX
MOV  BX,0107H         ;第一个数高 16位送 BX
ADC  BX,1006H         ;两个数的高 16位相加,结果送 BX
```

相加的最后结果为:110E21C8H。

3) 加 1 指令 INC

指令格式:

```
INC OPRD              ;OPRD←OPRD+1
```

INC 指令是将指定操作数的内容加 1,再送回该操作数。其操作类似于 C 语言中的"＋＋"运算符。这里,操作数 OPRD 可以是寄存器或存储器操作数;可以是 8 位,也可以是 16 位;但不能是段寄存器,也不能是立即数。例如:

```
INC AX                          ;AX←AX+1
INC BYTE PTR[SI]                ;将 SI内容为偏移地址的存储单元的内容+1,结果送回该单元
```

INC 指令不影响 CF 标志位,但对其他 5 个状态标志 AF、OF、PF、SF 及 ZF 会产生影响。它通常用于在循环程序中修改地址指针及循环次数等。

2. 减法指令

8088/8086 共有 5 条减法指令，它们是：不考虑借位的普通减法指令 SUB、考虑借位的减法指令 SBB、减 1 指令 DEC、求补指令 NEG 以及比较指令 CMP。

1）不考虑借位的减法指令 SUB

指令格式：

```
SUB OPRD1,OPRD2                       ;OPRD1←OPRD1-OPRD2
```

SUB 指令是一条双操作数指令，其功能是用目标操作数减去源操作数，并将结果送目标操作数所在地址中。

该指令对操作数的要求以及对状态标志位的影响与 ADD 指令完全相同。例如：

```
SUB BL,30H              ;BL←BL-30H
SUB AL,[BP+SI]          ;AL-SS:[BP+SI]单元内容,结果送 AL
```

2）考虑借位的减法指令 SBB

指令格式：

```
SBB   OPRD1,OPRD2       ;OPRD1←OPRD1-OPRD2-CF
```

SBB 指令的功能是用目标操作数减去源操作数以及标志位 CF 的值，并将结果送目标操作数所在的地址中。其对操作数的要求以及对状态标志位的影响与 SUB 指令完全相同。SBB 指令主要用于多字节的减法运算。例如：

```
SBB BL,30H              ;BL←BL-30H-CF
```

3）减 1 指令 DEC

指令格式：

```
DEC OPRD                ;OPRD←OPRD-1
```

DEC 指令与 INC 指令一样，是一条单字节指令，其功能是将操作数的值减 1，结果再送回该操作数所在地址。该指令对操作数的要求及对标志位的影响与 INC 指令相同。例如：

```
DEC AX                  ;AX←AX-1
DEC BYTE PTR[DI]        ;将数据段中 DI 所指单元的内容减 1,结果送回该单元中
```

DEC 指令常用于在循环程序中修改循环次数。

【例 3-23】 编写一个延时程序。

```
        MOV CX,0FFFFH   ;送计数初值到 CX
NEXT:   DEC CX          ;计数值 CX 减 1
        JNZ NEXT        ;若 CX≠0 则转 NEXT
        HLT             ;停止
```

4）求补指令 NEG

NEG 指令的操作是用 0 减去操作数 OPRD,结果送回该操作数所在地址。指令格式为

```
NEG OPRD                    ;OPRD←0-OPRD
```

操作数 OPRD 可以是寄存器或存储器操作数。利用该指令可以得到负数的绝对值。之所以把 NEG 指令称为求补指令,是因为对一个负数取补码就相当于用 0 减去此数。

例如:设 AL=FFH,执行指令 NEG AL 后,AL=0-FFH=01H,即实现了对 FFH(-1 的补码)求补,或说得到了 AL 中负数的绝对值。

NEG 指令对 6 个状态标志位均有影响。应用该指令时有以下两点需要注意:

(1) 执行 NEG 指令后,一般情况下都会使 CF 为 1。因为用 0 减去某个操作数,自然会产生借位,而减法的 CF 值正是反映无符号数运算中的借位情况。除非给定的操作数为 0 才会使 CF 为 0。

(2) 当指定的操作数的值为 80H(-128)或为 8000H(-32768),则执行 NEG 指令后结果不变,即仍为 80H 或 8000H,但 OF 置 1,其他情况下 OF 均置 0。

5) 比较指令 CMP

指令格式及操作:

```
CMP OPRD1,OPRD2             ;OPRD1-OPRD2,结果不送回 OPRD1
```

CMP 指令是用目标操作数减源操作数,但相减的结果不送回目标操作数,即指令执行后两操作数内容不变,而只是影响 6 个状态标志位。指令对操作数的要求及对标志位的影响与 SUB 指令完全相同。

比较指令主要是用来比较两个数的大小关系。可以在比较指令执行后根据标志位的状态判断两个操作数谁大谁小,或是否相等。判断方法如下。

(1) 相等关系。如果 ZF=1,则两个操作数相等;否则不相等。

(2) 大小关系。分无符号数和有符号数两种情况考虑。

① 对两个无符号数,根据 CF 标志位的状态确定。若 CF=0,则被减数大于减数(因为若被减数大于减数,则无须借位,即 CF=0)。

② 对两个有符号数,情况要稍微复杂一些,须考虑两个数是同符号还是异符号。因为有符号数用最高位来表示符号,可用 SF 来判断谁大谁小。

对两个同符号数,因相减不会产生溢出,即 OF=0,有:SF=0,被减数大于减数;SF=1,减数大于被减数。

如果比较的两个数符号不相同,此时就有可能出现溢出。

若 OF=0(即无溢出),则有:如果被减数大于减数,SF=0;如果被减数小于减数,SF=1;如果被减数等于减数,SF=0,同时 ZF=1。

若 OF=1(有溢出),则:如果被减数大于减数,SF=1;如果被减数小于减数,SF=0。

归纳以上结果,可得出判断两个有符号数大小关系的方法:当 OF⊕SF=0 时,被减数大于减数;当 OF⊕SF=1 时,减数大于被减数。

编程序时,一般在比较指令之后都紧跟一个条件转移指令,以根据比较结果决定程序的转向。

【例 3-24】 在内存数据段从 DATA 开始的单元中存放了两个 8 位无符号数,试比较

它们的大小,并将大的数送 MAX 单元。

```
        LEA BX,DATA         ;DATA 偏移地址送 BX
        MOV AL,[BX]         ;第一个无符号数送 AL
        INC BX              ;BX 加 1,指向第二个数
        CMP AL,[BX]         ;两个无符号数进行比较
        JNC DONE            ;若 CF=0(无进位,表示第一个数大),转向 DONE
        MOV AL,[BX]         ;否则,第二个无符号数送 AL
DONE:   MOV MAX,AL          ;将较大的无符号数送 MAX
        HLT                 ;停止
```

3. 乘法指令

乘法指令包括无符号数乘法和有符号数乘法指令两种,采用隐含寻址方式,隐含的目标操作数为 AX(与 DX),而源操作数由指令给出。指令可完成两个字节数相乘或字与字相乘。对 8 位数的乘法,乘积为 16 位,存放在 AX 中;对 16 位数相乘,乘积为 32 位,高 16 位放在 DX 中,低 16 位放在 AX 中。

无符号数乘法指令与有符号数乘法指令的区别主要表现在以下 3 个方面。

(1) 操作数的性质不同,前者是无符号数,后者要求两乘数都须为有符号数。

(2) 对无符号数乘法,如果乘积的高半部分(在字节相乘时为 AH,在字相乘时为 DX)不为 0,则 CF=OF=1,代表 AH 或 DX 中包含乘积的有效数字;否则 CF=OF=0。对有符号数乘法,若乘积的高半部分是低半部分的符号位的扩展,则 CF=OF=0;否则 CF=OF=1。对其他标志均无定义。

(3) 无符号数乘法指令中的源操作数应满足无符号数的表示范围,而有符号数乘法指令中给出的源操作数应满足带符号数的表示范围。

这里仅介绍无符号数的乘法指令 MUL,有关带符号数乘法指令 IMUL 的内容会在下面说明。

无符号数乘法指令的格式:

```
MUL OPRD
```

指令的操作为

```
字节乘法   AX←OPRD×AL
字乘法     DX:AX←OPRD×AX
```

这里,源操作数 OPRD 可以是 8 位或 16 位的寄存器或存储器。乘法指令要求两操作数字长相等,且不能为立即数。例如:

```
MUL BX                    ;DX:AX ← AX×BX
MUL BYTE PTR[SI]          ;AX ← AL×[SI]
MUL DL                    ;AX ← AL×DL
```

在某些情况下,可用左移指令来代替乘法指令以加快程序的运行速度。这点将在移

位指令中说明。

【例 3-25】 设 AL＝0FEH，CL＝11H，两数均为无符号数，求 AL 与 CL 的乘积。

```
MUL CL
```

指令执行后：AX＝10DEH，因 AH 中的结果不为零，故 CF＝OF＝1。

4. 除法指令

8088 的除法指令也包括无符号数的除法指令和有符号数的除法指令两种，同样采用隐含寻址方式，隐含了被除数，而除数由指令给出，要求除数不能为立即数。

除法指令要求被除数的字长必须为除数字长的两倍。若除数为 8 位，则被除数为 16 位，并放在 AX 中；若除数为 16 位，则被除数为 32 位，放在 DX 和 AX 中，其中 DX 放高 16 位，AX 放低 16 位。实际编程中，若被除数字长不够，就要使用 3.3.1 节介绍过的字位扩展指令来扩展其位数。

无符号数除法指令的格式为

```
DIV OPRD
```

指令中的操作数 OPRD 可以是 8 位或 16 位的寄存器或存储单元的内容。

指令的操作为

```
字节除法    AL←AX/OPRD, AH←AX % OPRD(%为取余数操作)
```

即 AX 中的 16 位无符号数除以 OPRD，得到的 8 位商放在 AL 中，8 位余数放在 AH 中。

```
字除法    AX←DX:AX/OPRD, DX←DX:AX % OPRD(%为取余数操作)
```

即 DX：AX 中的 32 位无符号数除以 OPRD，得到的 16 位商放在 AX 中，16 位余数放在 DX 中。

若除法运算的结果大于寄存器可保存的值，即超出了 8 位或 16 位无符号数的可表达范围，则在 CPU 内部会产生一个类型 0 中断。

例如：

```
DIV BL                ;AX除以 BL,商放 AL,余数放 AH
DIV WORD PTR[SI]      ;DX:AX除以 SI和 SI+1所指向单元的内容,商放 AX,余数放 DX
```

【例 3-26】 用除法指令计算 7FA2H÷03DDH。

```
MOV AX,7FA2H          ;AX=7FA2H
MOV BX,03DDH          ;BX=03DDH
CWD                   ;DX:AX=00007FA2H
DIV BX                ;商=AX=0021H,余数=DX=0025H
```

除法指令对 6 个标志位均无影响。

5. 其他算术运算指令

除以上指令外，8086 指令系统还具有其他一些算术运算指令，如表 3-3 所示。

表 3-3　其他算术运算指令

指令类型	汇编格式	指令的操作	示　　例
有符号数乘法指令	IMUL OPRD	字节乘法：AX ← OPRD×AL 字乘法：DX：AX ← OPRD ×AX	设 AL＝0FEH，CL＝11H，两操作数视为有符号数，则：IMUL CL；AX＝FFDEH＝－34。因 AH 中内容为 AL 中的符号扩展，故 CF＝OF＝0
有符号数除法指令	IDIV OPRD	功能和操作都和 DIV 指令类似，商和余数均为带符号数，且余数符号与被除数符号相同	IDIV CX ；DX 和 AX 中的 32 位数除以 CX，商在 AX 中，余数在 DX 中
BCD 码调整指令（需与相应的加、减、乘、除指令配合使用）	DAA	将按二进制运算规则执行后存放在 AL 中的结果调整为压缩 BCD 码	MOV AL，48H ADD AL，27H；AL＝6FH DAA；结果：AL＝75H
	AAA	对两个非压缩（扩展）BCD 数相加之后存放于 AL 中的和进行调整，形成正确的扩展 BCD 码，调整后的结果的低位在 AL 中，高位在 AH 中	MOV AL，09H ADD AL，4 AAA；结果：AL＝03H AH＝1，CF＝1
	DAS	对两个压缩 BCD 码相减后的结果（在 AL 中）进行调整，产生正确的压缩 BCD 码	
	AAS	对两个非压缩 BCD 码数相减之后的结果（在 AL 中）进行调整，形成一个正确的非压缩 BCD 码，其低位在 AL 中，高位在 AH 中	
	AAM	对两个非压缩 BCD 数相乘的结果（AX 中）进行调整，得到正确的非压缩 BCD 数（把 AL 寄存器的内容除以 OAH，商放 AH 中，余数放 AL 中）	MOV AL，07H MOV BL，09H MUL BL；AX＝003FH AAM；结果： AX＝0603H， 即非压缩 BCD 数 63
	AAD	在进行除法之前执行。将 AX 中的非压缩 BCD 码（十位数放 AH，个位数放 AL）调整为二进制数，并将结果放 AL 中	MOV AX，0203H；AX＝23 MOV BL，4 AAD；AX＝0017H DIV BL；结果： AH＝03H，AL＝05H

注：BCD 码调整指令仅对部分状态标志位有影响。

3.3.3 逻辑运算和移位指令

逻辑运算和移位指令包括逻辑运算指令和移位指令两大部分,移位指令中又分为非循环移位指令和循环移位指令。

1. 逻辑运算指令

8088/8086 提供的逻辑运算指令共有 5 条,包括 AND(逻辑“与”)、OR(逻辑“或”)、NOT(逻辑“非”)、XOR(逻辑“异或”)及 TEST(测试)指令。这些指令可对 8 位或 16 位的寄存器或存储器单元中的内容进行按位操作。除 NOT 指令外,其他 4 条指令对操作数的要求与 MOV 指令相同。它们的执行都会使 CF=OF=0,AF 值不定,并对 SF、PF 和 ZF 有影响。NOT 指令对操作数的要求与 INC 指令相同,但其执行对所有标志位都不影响。

1) 逻辑“与”指令 AND

指令格式:

```
AND OPRD1,OPRD2            ;OPRD1←OPRD1∧OPRD2
```

AND 指令使源操作数和目标操作数按位相“与”,结果送回目标操作数中。AND 指令在程序中主要应用于 3 个方面。

(1) 实现两操作数按位相“与”。例如:

```
AND AX,[BX]                ;AX和[BX]所指字单元的内容按位相"与",结果送 AX
```

(2) 使目标操作数中某些位保持不变,把其他位清 0。例如:

```
AND AL,0FH                 ;将 AL的高 4位清 0,低 4位保持不变
```

此时需要指定一个屏蔽字,屏蔽字各位的设置原则是:目标操作数中哪些位要清 0,就把屏蔽字相对应的位设为 0,其他位设为 1。如上例中,0FH 就是屏蔽字,其高 4 位为 0,低 4 位为 1,表示将 AL 中的高 4 位清除,而低 4 位保留。

(3) 使操作数不变,但影响 6 个状态标志位,并使 CF=OF=0。例如:

```
AND AX,AX                  ;AX自身按位相"与",不改变 AX内容,但影响 6个状态标志位
```

2) 逻辑“或”指令 OR

指令格式:

```
OR OPRD1,OPRD2             ;OPRD1←OPRD1∨OPRD2
```

OR 指令实现对源操作数和目标操作数按位相“或”,结果送回目标操作数中。对应 AND 指令,OR 指令在程序中也主要应用于以下 3 个方面。

(1) 实现两操作数按位相“或”。例如:

```
OR [BX],AL                 ;[BX]单元的内容和 AL的内容相"或",结果送回 [BX]单元
```

(2) 使目标操作数某些位保持不变,将另外一些位置1。此时源操作数应这样设置:目标操作数哪些位需要置为“1”,就把源操作数中与之对应的位设为1,其他位设为0。例如:

```
OR AL,20H                ;将 AL 中的 D5 位置 1,其余位不变
```

(3) 使操作数不变,但影响6个状态标志位,并使CF=OF=0。例如:

```
OR AX,AX                 ;AX 内容不变,但影响 6 个状态标志位
```

【例3-27】 为了保证数据通信的可靠性,往往需要对传送的ASCII码数据进行校验。校验的方法之一就是使用奇偶校验,偶校验是使要传送的ASCII码中1的个数为偶数,奇校验则使1的个数为奇数。奇偶校验位放在ASCII码的最高位上。

假定要传送的ASCII码在AL中,则对AL的内容加上偶校验的程序段如下。

```
          OR AL,AL                 ;不改变 AL 中的内容,但影响各标志位
          JPE CONTINUE             ;若 PF=1(AL 中 1 的个数为偶数)则转移
          OR AL,80H                ;若 AL 中 1 的个数为奇数则将其变为偶数
CONTINUE: …
```

3) 逻辑“非”指令NOT

指令格式:

```
NOT OPRD
```

NOT指令是单操作数指令,它将指定的操作数OPRD按位取反,再送回该操作数。这里,OPRD可以是8位或16位的寄存器或存储器操作数,但不能是立即数。NOT指令对标志位无影响。

例如:

```
NOT AX                   ;将 AX 中内容按位取反,结果送回 AX
NOT WORD PTR[SI]         ;将[SI]所指两个单元中的内容按位取反,再送回这两单元
```

4) 逻辑“异或”指令XOR

指令格式及操作:

```
XOR OPRD1,OPRD2          ;OPRD1←OPRD1⊕OPRD2
```

XOR指令将源操作数和目标操作数按位进行“异或”运算,结果送回目标操作数。“异或”操作的原则是:两位操作数相同时结果为0,不同时结果为1。例如:

```
XOR AX,1122H             ;AX 的内容与 1122H"异或",结果送回 AX
```

根据“异或”运算的性质,某一操作数和自身相“异或”,结果为零。在程序中常利用这一特性,使某寄存器清零。例如:

```
XOR AX,AX                ;使 AX 清零
```

5) 测试指令TEST

TEST指令的格式、对操作数的要求及完成的操作和AND指令类似,区别是:

TEST 指令不将“与”的结果送回目标操作数，而只影响标志位，故这条指令常用于在不破坏目标操作数内容的情况下检测操作数中某些位是“1”还是“0”。例如：

```
TEST AL,02H                       ;若 AL 中 D1 位为 1,则 ZF=0,否则 ZF=1
```

【例 3-28】 从 4000H 开始的单元中放有 32 个有符号数，要求统计出其中负数的个数，并将统计结果存入 BUFFER 字单元中(即：BUFFER 为 16 位存储器操作数)。程序段如下：

```
        XOR DX,DX                 ;清 DX 内容,DX 用于存放中间结果
        MOV SI,4000H              ;SI←起始地址
        MOV CX,20H                ;CX←统计次数
AGAIN:  MOV AL,[SI]               ;AL←取第一个数
        INC SI                    ;地址指针加 1
        TEST AL,80H               ;测试所取的数是否为负数
        JZ NEXT                   ;不为负数则转 NEXT
        INC DX                    ;若为负数则 DX←DX+1
NEXT:   DEC CX                    ;CX←CX-1
        JNZ AGAIN                 ;若 CX≠0 则继续检测下一个
        MOV BUFFER,DX             ;统计结果送 BUFFER 单元
```

2. 移位指令

移位指令包括非循环移位指令和循环移位指令两类。指令实现将寄存器操作数或内存操作数进行指定次数的移位。当移动一位时，移动次数由指令直接给出，在移 2 位或更多位时，移动的位数要放在 CL 寄存器中，即指令的原操作数是移位次数(1 或 CL)，目标操作数是被移动的对象(8 位或 16 位的寄存器或存储器单元)。这类指令的执行大多会影响 6 个状态标志位。

1) 非循环移位指令

8086/8088 有 4 条非循环移位指令，分别是：算术左移指令 SAL(Shift Arithmetic Left)、算术右移指令 SAR(Shift Arithmetic Right)、逻辑左移指令 SHL(Shift Logic Left)、逻辑右移指令 SHR(Shift Logic Right)。

4 条指令的格式完全相同，可实现对 8 位或 16 位寄存器操作数或内存操作数进行指定次数的移位。逻辑移位指令针对的是无符号数，算术移位指令针对有符号数。

(1) 算术左移和逻辑左移指令 SAL/SHL。算术左移指令 SAL 和逻辑左移指令 SHL 执行完全相同的操作，其指令格式为

```
SHL OPRD,1        SAL OPRD,1
```

或

```
SHL OPRD,CL        SAL OPRD,CL
```

SHL/SAL 指令执行的操作是将目的操作数的内容左移一位或 CL 所指定的位，每左移一位，左边的最高位移入标志位 CF，而在右边的最低位补零。指令操作的示意图如图 3-15 所示。

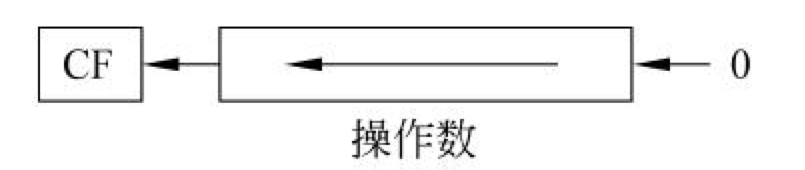


图 3-15 SHL/SAL 左移指令操作示意图

在移动次数为 1 的情况下，若移位之后操作数的最高位与 CF 标志位状态不相同，则 OF=1；否则 OF=0。这可用于判断移位前后的符号位是否一致。另外，指令还影响标志位 PF、SF 和 ZF。

OF=1 对 SHL 指令不表示左移后溢出，而对 SAL 指令则表示移位后超出了符号数的表示范围。

【例 3-29】

```
MOV AL,41H
SHL AL,1
```

执行结果为 AL=82H，CF=0，OF=1。若视 82H 为无符号数，则它没有溢出(82H<FFH)；若视它为有符号数，则溢出了(82H>7FH)，因为移位后正数变成了负数。

将一个二进制数无符号数左移一位相当于将该数乘 2，所以可利用左移指令实现把一个数乘上 2^i 的运算。由于左移指令比乘法指令的执行速度快得多，在程序中用左移指令来代替乘法指令可加快程序的运行。

【例 3-30】 把以 DATA 为首址的两个连续单元中的 16 位无符号数乘以 10。

因为 $10x=8x+2x=2^3x+2^1x$，所以可用左移指令实现该乘法运算。程序如下：

```
LEA SI,DATA            ;DATA单元的偏移地址送 SI
MOV AX,[SI]            ;AX←被乘数
SHL AX,1               ;AX=DATA*2
MOV BX,AX              ;暂存 BX
MOV CL,2               ;CL←移位次数
SHL AX,CL              ;AX=DATA*8
ADD AX,BX              ;AX=DATA*10
HLT
```

(2) 逻辑右移指令 SHR。SHR 指令格式与 SHL 相同，它将指令中的目标操作数视为无符号数。其操作是将目标操作数顺序向右移一位或 CL 指定的位数，每右移一位，右边的最低位移入标志位 CF，而在左边的最高位补零。指令操作的示意图如图 3-16 所示。

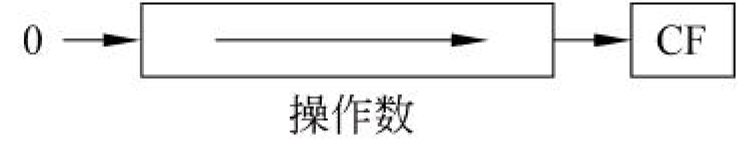


图 3-16 SHR 指令操作示意图

SHR 指令也影响标志位 CF 和 OF。如果移动次数为 1 且移位之后新的最高位和次高位不相等，则标志位 OF=1；否则 OF=0。若移位次数不为 1，则 OF 状态不定。

【例 3-31】

```
MOV AL,82H
SHR AL,1
```

执行结果：AL=41H，CF=0，OF=1。

与左移类似，每逻辑右移一位，相当于无符号的目标操作数除以 2。因此同样可利用 SHR 指令完成把一个数除以 2^i 的运算。SHR 指令的执行速度也比除法指令要快得多。

(3) 算术右移指令 SAR。SAR 指令是将指令中目标操作数视为有符号数，格式与

SHR 相同。指令的操作是将目标操作数顺序向右移一位或 CL 指定的位数，操作数最低位移入标志位 CF。它与 SHR 指令的区别是：算术右移时最高位不是补零，而是保持不变。指令的操作如图 3-17 所示。将例 3-31 中的 SHR 指令改为 SAR 指令，则指令的执行结果为：AL＝C1H，CF＝0。



图 3-17 SAR 指令操作示意图

SAR 指令对标志位 CF、PF、SF 和 ZF 有影响，但不影响 OF、AF。

同样，算术右移指令也可以完成有符号操作数除以 2^i 的运算。

2）循环移位指令

8088 CPU 有 4 条循环移位指令，它们是：不带进位标志位 CF 的循环左移指令 ROL、不带进位标志位 CF 的循环右移指令 ROR、带进位标志位 CF 的循环左移指令 RCL、带进位标志位 CF 的循环右移指令 RCR。

循环移位指令的操作数类型及指令格式与非循环移位指令相同。4 条指令的操作示意图如图 3-18 所示。

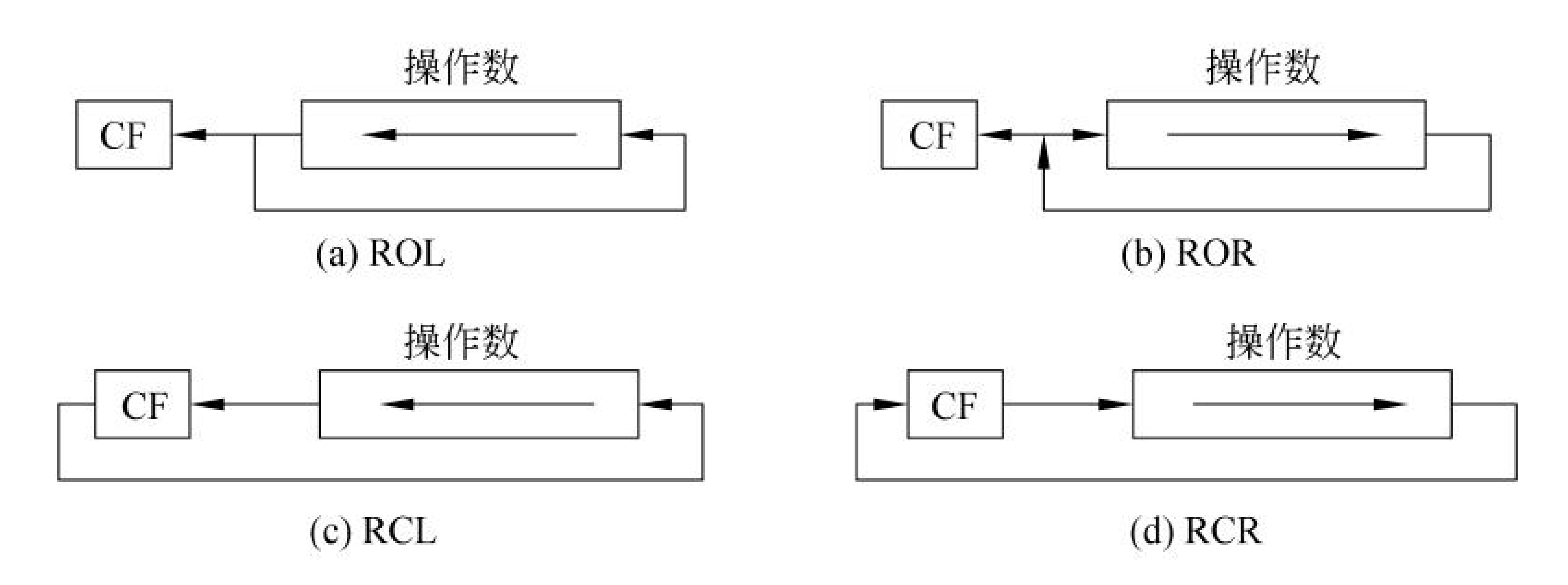


图 3-18 循环移位指令操作示意图

（1）不带 CF 的循环左移指令 ROL。

指令格式：

```
ROL OPRD,1
```

或

```
ROL OPRD,CL
```

ROL 指令将目标操作数向左循环移动一位或由 CL 指定的位数，最高位移入 CF，同时再移入最低位构成循环，进位标志 CF 不在循环之内如图 3-18(a)所示。

ROL 指令影响标志位 CF 和 OF。若循环移位次数为 1，且移位之后目标操作数的最高位和 CF 值不相等，则标志位 OF＝1，否则 OF＝0；若移位次数不为 1，OF 状态不定。

【例 3-32】

```
MOV AL,82H
ROL AL,1
```

执行结果：AL＝05H，CF＝1，OF＝1。

(2) 不带 CF 的循环右移指令 ROR。

指令格式：

```
ROR OPRD,1
```

或

```
ROR OPRD,CL
```

ROR 指令将目标操作数向右循环移动一位或 CL 指定位数，最低位移入 CF，同时再移入最高位构成循环，如图 3-18(b)所示。

ROR 指令影响标志位 CF 和 OF。如果循环移位次数为 1，且移位之后新的最高位和次高位不等，则标志位 OF=1，否则 OF=0；若移位次数不为 1，则 OF 状态不定。

若将例 3-32 中的 ROL 指令改为 ROR 指令，则指令的执行结果为：AL=41H，CF=0，OF=1。

(3) 带 CF 的循环左移指令 RCL。

指令格式 ：

```
RCL OPRD,1
```

或

```
RCL OPRD,CL
```

RCL 指令将目标操作数连同进位标志位 CF 一起向左循环移动一位或 CL 指定位数，最高位移入 CF，而 CF 原来的值移入最低位，如图 3-18(c)所示。

RCL 指令对标志位的影响与 ROL 指令相同。

【例 3-33】

```
RCL BYTE PTR[100AH],1
```

设 DS=6000H，且指令执行前[6100AH]=8EH，CF=0。

执行上面的指令后：[6100AH]=1CH，CF=1，操作示意图如图 3-19 所示。

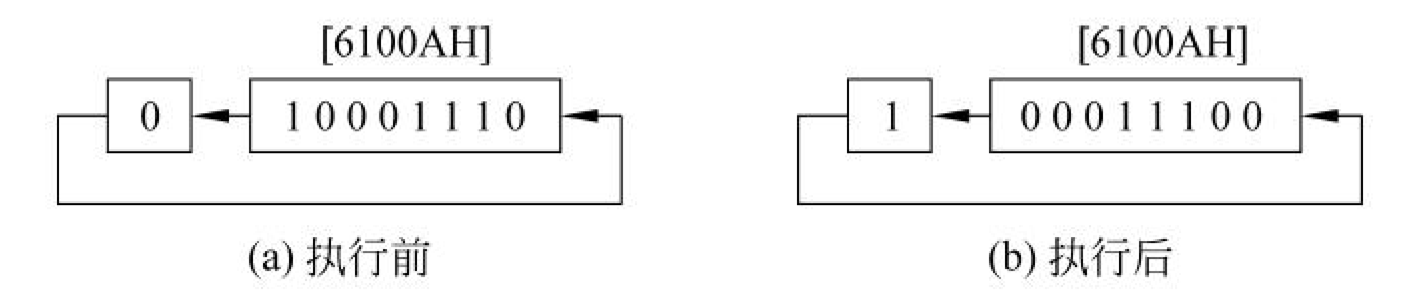


图 3-19　RCL 指令执行举例

(4) 带 CF 的循环右移指令 RCR。

指令格式：

```
RCR OPRD,1
```

或

```
RCR OPRD,CL
```

RCR 指令将目标操作数连同进位标志位 CF 一起向右循环移动一位或 CL 指定位数，最低位移入 CF，而 CF 原来的值移入最高位，如图 3-18(d)所示。

RCR 指令对标志位的影响与 ROR 指令相同。

循环移位指令与非循环移位指令不同，循环移位后，操作数中原来各位数的信息不会丢失，而只是改变了位置而已(仍在操作数中的其他位置上或 CF 中)，如果需要还可恢复(反向移动即可)。

利用循环移位指令可以测试操作数某一位的状态。

【例 3-34】 测试 BL 寄存器中第 4 位的状态，并保持原内容不变。程序如下：

```
        MOV CL,4                    ;CL←移位次数
        ROL BL,CL                   ;CF←BL第 4位
        JNC ZERO                    ;如果 CF=0则转到 ZERO
        ROR BL,CL                   ;恢复原 BL内容
        …
ZERO:   ROR BL,CL                   ;恢复原 BL内容
        …
```

例 3-34 显然也可用 TEST 指令来实现，具体的程序编写留作读者思考。

【例 3-35】 将 DX 和 AX 两个寄存器组合成一个 32 位操作数，一起逻辑左移一位，即 AX 的最高位应移入 DX 的最低位，如图 3-20 所示，可用两条指令实现这个操作。

```
SHL AX,1              ;AX左移一位,CF←AX最高位
RCL DX,1              ;DX带进位位循环左移一位,DX最低位←CF
```

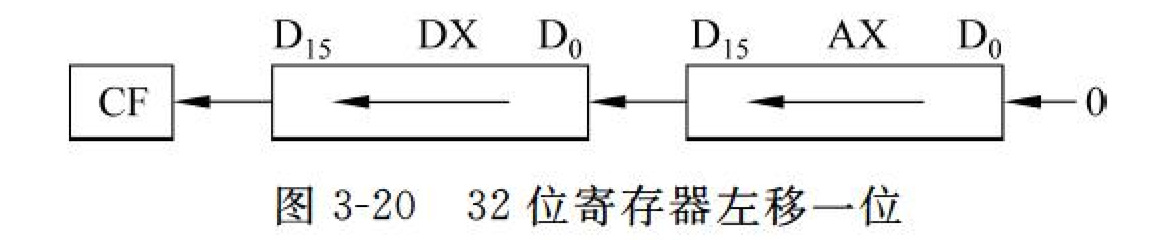


图 3-20 32 位寄存器左移一位

3.3.4 串操作指令

1. 串操作指令的共同特点

人们将存储器中的地址连续的若干单元的字符或数据称为字符串或数据串。串操作指令就是用来对串中每个字符或数据做同样操作的指令。串指令既可处理字节串，也可处理字串，并在每完成一个字节(或字)的操作后能够自动修改指针，去执行下一个字节(或字)的操作。8086 指令集中的串操作指令可以处理的最大串长度为 64KB。

所有的串操作指令(除与累加器打交道的串操作指令外)都具有以下共同点。

(1) 源串(源操作数)默认为数据段，即段基地址在 DS 中，但允许段重设。偏移地址用 SI 寄存器指定，即源串指针为 DS:SI。

(2) 目标串(目标操作数)默认在 ES 附加段中，不允许段重设。偏移地址用 DI 寄存器指定，即目标串指针为 ES:DI。

(3) 串长度值放在 CX 寄存器中。

(4) 串操作指令本身可实现地址指针的自动修改。在对每个字节(或字)操作后,SI 和 DI 寄存器的内容会自动修改,修改方向与标志位 DF 的状态有关。若 DF=0,SI 和 DI 按地址增量方向修改(对字节操作加 1,对字操作则加 2);否则,SI 和 DI 按地址减量方向修改。

(5) 可以在串操作指令前使用重复前缀。若使用了重复前缀,在每一次串操作后,CX 的内容会自动减 1。

综上所述,使用串操作指令关键的要点是:应预先设置源串指针(DS、SI)、目标串指针(ES、DI)、重复次数(CX)以及操作方向(DF)。

2. 重复操作前缀

在串操作指令前面加一个适当的重复操作前缀,能够使该指令重复执行,即指令在执行时不仅能够按照 DF 所决定的方向自动修改地址指针 SI 和 DI 的内容,还可在每完成一次操作后自动修改串长度 CX 的值,重复执行串指令,直至 CX=0 或满足指定的条件为止。

用于串操作指令的重复操作前缀分为两类:无条件重复前缀(1 条)及有条件重复前缀(共 4 条),它们分别如下。

(1) REP:无条件重复前缀,重复执行指令规定的操作,直到(CX)=0。

(2) REPE/REPZ:相等/结果为零时重复,ZF=1,且 CX≠0 时重复。

(3) REPNE/REPNZ:不相等/结果不为零时重复,ZF=0,且 CX≠0 时重复。

加重复操作前缀可简化程序的编写,并加快串运算指令的执行速度。加重复操作前缀之后的串操作指令的执行动作可表示为:①执行规定的操作;②SI 和 DI 自动增量(或减量);③CX 内容自动减 1;④根据 ZF 的状态自动决定是否重复执行。

3. 串操作指令

串操作指令是 8086 指令系统中唯一一组能直接处理源和目标操作数都在存储单元的指令。串操作指令共有 5 条。

1) 串传送指令

串传送指令(Move String)有 3 种指令格式:

```
MOVS OPRD1,OPRD2
MOVSB
MOVSW
```

第一种格式中,OPRD1 为目标串地址,OPRD2 为源串地址。指令将源串地址中的字节或字传送到目标串地址中。源串和目标串的段地址可以使用默认值(即预先对 DS、ES 设定的值),源串也可用段重设指定在其他段中。第一种格式多用于需要段重设的情况。第二种和第三种格式隐含了两个操作数的地址,此时源串和目标串地址必须符合默认值,即源串在数据段、偏移地址在 SI 中,目标串在附加段、偏移地址在 DI 中。

MOVSB 指令一次完成一个字节的传送，MOVSW 一次完成一个字的传送。

串传送指令实现内存单元到内存单元的数据传送，解决了 MOV 指令不能直接在内存单元之间传送数据的限制。

MOVS 指令常与无条件重复前缀 REP 联合使用，以提高程序运行速度。

【例 3-36】 将 2000H:1200H 地址开始的 100 个字节传送到 6000H:0000H 开始的内存单元中去。程序如下：

```
MOV AX,2000H
MOV DS,AX                ;设定源串段地址
MOV AX,6000H
MOV ES,AX                ;设定目标串段地址
MOV SI,1200H             ;设定源串偏移地址
MOV DI,0                 ;设定目标串偏移地址
MOV CX,100               ;串长度送 CX
CLD                      ;DF=0,使地址指针按增量方向修改
REP MOVSB                ;每传送一个字节,自动修改地址指针及 CX 直至 CX=0
HLT                      ;暂停执行
```

串传送指令的执行不影响标志位。

2）串比较指令

串比较指令(Compare String)有 3 种格式：

```
CMPS  OPRD1,OPRD2
CMPSB
CMPSW
```

串比较指令与比较指令 CMP 的操作类似，CMP 指令比较的是两个数据，而 CMPS 进行的是两个数据串的比较。它将源串地址与目标串地址中的数据串按字节(或字)进行比较，比较结果不送回目标串地址中，而只反映在标志位上。每进行一次比较后自动修改地址指针，指向串中的下一个元素。在以上 3 种格式中，第一种格式主要用在需要段重设的情况下。CMPSB 是按字节进行比较，CMPSW 是按字进行比较。

串比较指令通常和条件重复前缀 REPE(REPZ)或 REPNE(REPNZ)连用，用来检查两个字符串是否相等。

在加条件重复前缀的情况下，结束串比较指令的执行就有两种可能：①不满足条件前缀所要求的条件；②CX＝0(此时表示已全部比较结束)。因此在程序中，串比较指令的后边需要一条指令来判断是何种原因结束了串比较，判断的条件是 ZF 标志位。串比较指令的执行会影响 ZF 的状态。对 REPE/REPZ，ZF＝1 会重复；对 REPNE/REPNZ，ZF＝0 则会重复。CX 是否为零不影响 ZF 状态。

【例 3-37】 比较两个字符串是否相同，并找出其中第一个不相等字符的地址，将该地址送 BX，不相等的字符送 AL。两个字符串的长度均为 200 个字节，M1 为源串首地址，M2 为目标串首地址。程序如下：

```
        LEA SI,M1               ;SI←源串首地址
        LEA DI,M2               ;DI←目标串首地址
        MOV CX,200              ;CX←串长度
        CLD                     ;DF=0,使地址指针按增量方向修改
        REPE CMPSB              ;若相等则重复比较
        JZ STOP                 ;若 ZF=1,表示两数据串完全相等,转 STOP
        DEC SI                  ;否则 SI-1,指向不相等单元
        MOV BX,SI               ;BX←不相等单元的地址
        MOV AL,[SI]             ;AL←不相等单元的内容
STOP:   HLT                     ;停止
```

程序中找到第一个不相等字符后，地址指针自动加 1，所以将地址指针再减 1 即可得到不相等单元的地址。

3）串扫描指令

串扫描指令(Scan String)有 3 种格式：

```
SCAS  OPRD          ;OPRD为目标串
SCASB
SCASW
```

SCAS 指令的执行与 CMPS 指令类似，也是进行比较操作。只是 SCAS 指令是用累加器 AL 或 AX 的值与目标串(由 ES:DI 指定)中的字节或字进行比较，比较结果不改变目标操作数，只影响标志位。

SCAS 指令常用来在一个字符串中搜索特定的关键字，把要找的关键字放在 AL(或 AX)中，再用本指令与字符串中各字符逐一比较。

【例 3-38】 在 ES 段中从 2000H 单元开始存放了 10 个字符，寻找其中有无字符“A”。若有则记下搜索次数(次数放 DATA1 单元)，并记下存放“A”的地址(地址放 DATA2 单元)。程序段如下：

```
        MOV DI,2000H        ;目标字符串首地址送 DI
        MOV BX,DI           ;首地址暂存在 BX 中
        MOV CX,0AH          ;字符串长度送 CX
        MOV AL,'A'          ;关键字"A"的 ASCII 码送 AL
        CLD                 ;清 DF,每次扫描后指针增量
        REPNZ SCASB         ;扫描字符串,直到找到"A"或 CX=0
        JZ FOUND            ;若找到则转移
        MOV DI,0            ;没找到要搜索的关键字,使 DI=0
        JMP DONE
FOUND:  DEC DI              ;DI-1,指向找到的关键字所在地址
        MOV DATA2,DI        ;将关键字地址送 DATA2 单元
        INC DI
        SUB DI,BX           ;用找到的关键字地址减去首地址得到搜索次数
DONE:   MOV DATA1,DI        ;将搜索次数送 DATA1 单元
        …
```

上面的程序中,SCAS 指令加上前缀 REPNZ 表示串元素不等于关键字(ZF＝0)且串未结束(CX≠0)时就继续搜索。若此例改为找到第一个不是“A”的字符,则 SCAS 前应加上前缀 REPZ,表示串元素等于关键字且串未结束时就继续搜索。

例 3-38 中,退出 REPNZ SCASB 串循环有两种可能:①已找到关键字,从而退出,此时 ZF＝1;②未搜索到关键字,但串已检索完毕,从而退出,此时 ZF＝0,CX＝0。因而退出之后,可根据对 ZF 标志的检测来判断是属于哪种情况。

同例 3-37 一样,执行 REPNZ SCASB 操作时,每比较一次,目的串指针自动加 1(因 DF＝0),所以找到关键字后,需将 DI 内容减 1 才能得到关键字所在地址。

4) 串装入指令

串装入指令(Load String)有 3 种格式:

```
LODS  OPRD                    ;OPRD为源串
LODSB
LODSW
```

LODS 指令把由 DS:SI 指向的源串中的字节或字取到累加器 AL 或 AX 中,并在这之后根据 DF 的值自动修改指针 SI,以指向下一个要装入的字节或字。

LODS 指令不影响标志位且一般不带重复前缀,因为每重复一次 AL 或 AX 中内容将被后一次所装入的字符所取代。

【例 3-39】 以 MEM 为首地址的内存区域中有 10 个以非压缩 BCD 码形式存放的十进制数,它们的值可能是 0～9 中的任意一个,现编程序将这 10 个数顺序显示在屏幕上。程序段如下:

```
        LEA SI,MEM            ;SI←源串偏移地址
        MOV CX,10             ;设置串长度
        CLD                   ;DF←0
        MOV AH,02H            ;AH←功能号(表示单字符显示输出)
NEXT:   LODSB                 ;取一个 BCD 码到 AL
        ADD AL,30H            ;BCD 码转换为对应的 ASCII 码
        MOV DL,AL             ;DL←字符的 ASCII 码
        INT 21H               ;输出显示
        DEC CX                ;CX←CX-1
        JNZ NEXT              ;ZF=0 则重复
        HLT
```

LODSB 指令可用来代替以下 2 条指令:

```
MOV  AL,[SI]
INC  SI
```

而 LODSW 指令可用来代替以下 3 条指令:

```
MOV  AX,[SI]
INC  SI
INC  SI
```

5）串存储指令

串存储指令(Store String)有 3 种格式：

```
STOS OPRD                    ;OPRD为目标串
STOSB
STOSW
```

STOS 指令把累加器 AL 中的字节或 AX 中的字存到由 ES:DI 指向的存储器单元中，并在这之后根据 DF 的值自动修改指针 DI 的值(增量或减量)，以指向下一个存储单元。利用重复前缀 REP 可对连续的存储单元存入相同的值。指令对标志位没有影响。

【例 3-40】 把 6000H:1200H 单元开始的 100 个字存储单元内容清零，可用串存储指令实现。程序如下：

```
MOV AX,6000H
MOV ES,AX                    ;ES←目标串的段地址
MOV DI,1200H                 ;DI←目标串的偏移地址
MOV CX,100                   ;CX←串长度
CLD                          ;DF←0,从低地址到高地址的方向进行存储
MOV AX,0                     ;AX←0,即要存入到目的串的内容
REP STOSW                    ;将 100个单元清零
HLT
```

3.3.5 程序控制指令

程序控制指令包括转移指令、循环控制指令、过程调用指令和中断控制指令四大类，用于程序的分支转移、循环控制及过程调用等。

1. 无条件转移指令 JMP

JMP(Jump)指令的操作是无条件地使程序转移到指定的目标地址，并从该地址开始执行新的程序段。寻找目标地址的方法有两种：①直接的方式；②间接的方式。另外，考虑到 8086/8088 的内存是分段管理，因此将无条件转移指令分成 4 种。

1）段内直接转移

指令格式：

```
JMP LABEL
```

这里，LABEL 是一个标号，也称为符号地址，它表示转移的目的地。该标号在本程序所在代码段内。指令被汇编时，汇编程序会计算出 JMP 指令的下一条指令到 LABEL 所指示的目标地址之间的位移量(也就是相距多少个字节单元)，该地址位移量可正可负，可以是 8 位的或 16 位的。若为 8 位，表示转移范围为 −128～+127 字节；若位移量为 16 位，表示转移范围为 −32768～+32767 字节。段内转移时的标号前可加运算符 NEAR，也可不加，省略时为段内转移。

指令的操作是将 IP 的当前值加上计算出的地址位移量形成新的 IP，并使 CS 保持不变，从而使程序按新地址继续运行(即实现了程序的转移)。

【例 3-41】

```
        ⋮
        MOV AX,BX
        JMP NEXT                    ;无条件段内转移，转向符号地址 NEXT 处
        AND CL,0FH
        ⋮
NEXT:   OR CL,7FH
```

这里，NEXT 是一个段内标号，汇编程序计算出 JMP 的下条指令(即 AND CL，0FH)的地址到 NEXT 标号代表的地址之间的距离(也就是相对位移量)。执行 JMP 指令时，将这个位移量加到 IP 上，于是在执行完 JMP 指令后不再执行 AND CL，7FH 指令(因为 IP 已经改变)，而转去执行 OR CL，7FH 指令(因为此时 IP 指向这条指令)。

2) 段内间接转移

指令格式：

```
JMP OPRD
```

指令中的操作数 OPRD 是 16 位的寄存器或者存储器地址，可以采用各种寻址方式。指令的执行是用指定的 16 位寄存器内容或存储器两单元内容作为转移目标的偏移地址，用其内容取代原来 IP 的内容，从而实现程序的转移。

例如：

```
JMP BX                      ;指令执行后，IP=BX
JMP WORD PTR[BX+ DI]
```

对上面第二条指令，设指令执行前：DS＝3000H，BX＝1300H，DI＝1200H，[32500H]＝2350H，则指令执行后，IP＝2350H。指令执行的过程如图 3-21 所示。

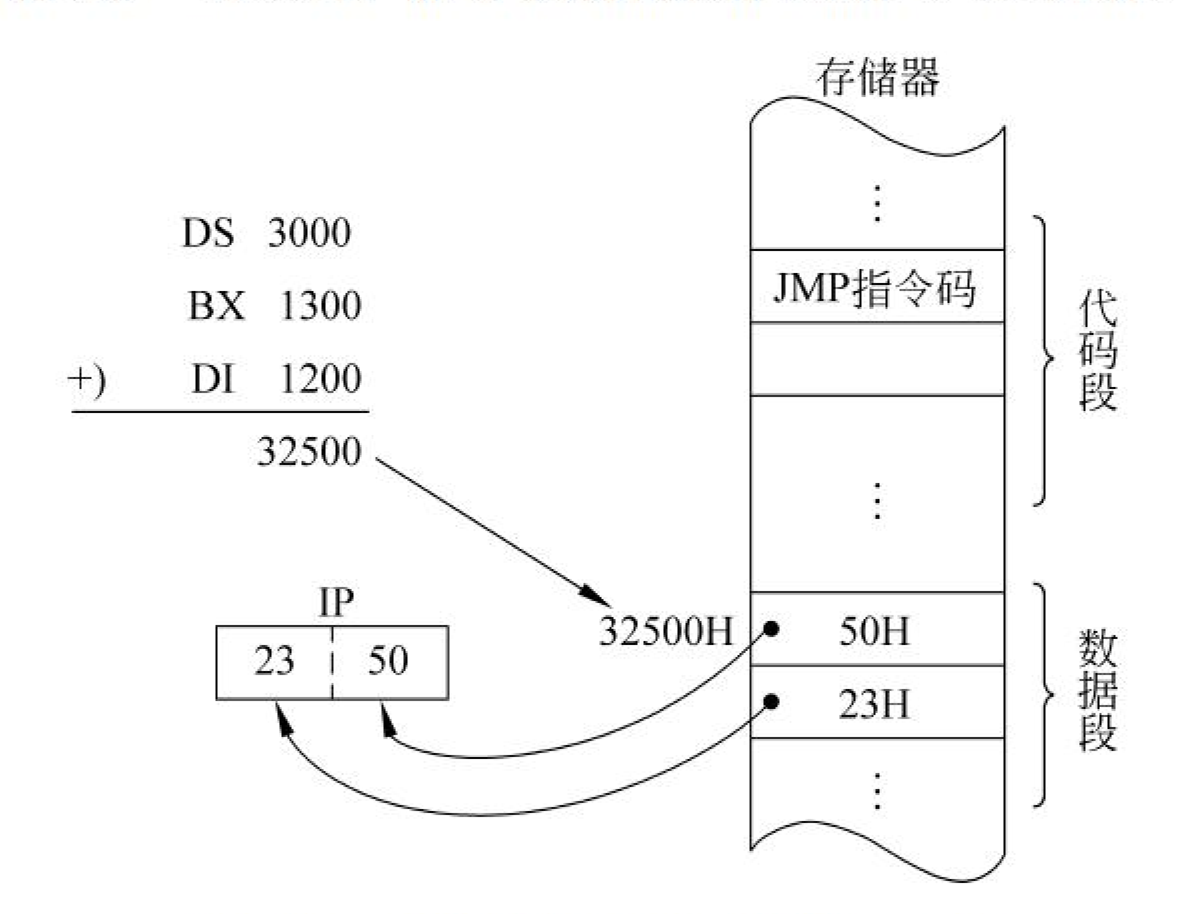


图 3-21 段内间接转移指令操作示意图

在上述指令中，若操作数 OPRD 是存储器，则要加上类型指示符 WORD PTR 以说明后边的存储器操作数是一个字(因为要送到 IP 的偏移地址是 16 位的)。另外，由于是段内转移，其范围一定在当前代码段内，所以 CS 的内容不变。

3）段间直接转移

采用段间直接转移时，指令中直接提供了要转移的 16 位段地址和 16 位偏移地址。

指令格式：

```
JMP   FAR LABEL
```

这里，FAR 表明其后的标号 LABEL 是一个远标号，即它在另一个代码段内。汇编程序根据 LABEL 的位置确定出 LABEL 所在的段基地址和偏移地址，然后将段地址送入 CS，偏移地址送入 IP，结果使程序转移到另一个代码段(CS:IP)继续执行。例如：

```
JMP FAR PTR NEXT                    ;远转移到 NEXT 处
JMP 8000H:1200H                     ;IP ← 1200H,CS←8000H
```

4）段间间接转移

指令格式：

```
JMP OPRD
```

这里，操作数 OPRD 是一个 32 位的存储器地址。指令的执行是将指定的连续 4 个内存单元的内容送入 IP 和 CS 中(低字内容送 IP，高字内容送 CS)，从而程序转移到另一个代码段继续执行。此处的存储单元地址可采用 3.2 节讲过的各种寻址方式(立即数和寄存器方式除外)。

【例 3-42】

```
JMP DWORD PTR[BX]
```

设指令执行前：DS＝3000H，BX＝3000H，[33000H]＝0BH，[33001H]＝20H，[33002H]＝10H，[33003H]＝80H，则指令执行后，IP＝200BH，CS＝8010H。

转移的目标地址＝8210BH。其操作示意图如图 3-22 所示。

由于段间转移是控制程序转移到另一个代码段中，不仅 IP 的内容要改变，CS 的内容也要改变，即转移地址一定是 32 位字长。因此，在操作数前要加上 DWORD PTR，表示其后的操作数为双字。

JMP 指令对标志位无影响。

2. 条件转移指令 JCC

8088/8086 共有 18 条不同的条件转移指令，如表 3-4 所示。它们根据其前一条指令执行后标志位的状态来决定程序是否转移。若满足转移指令所规定的条件，则程序转移到指令指定的地址去执行从那里开始的指令；若不满足条件，则顺序执行下一条指令。所有的条件转移都是直接寻址方式的短转移，即只能在以当前 IP 值为中心的－128～＋127 字节范围内转移。条件转移指令不影响标志位。

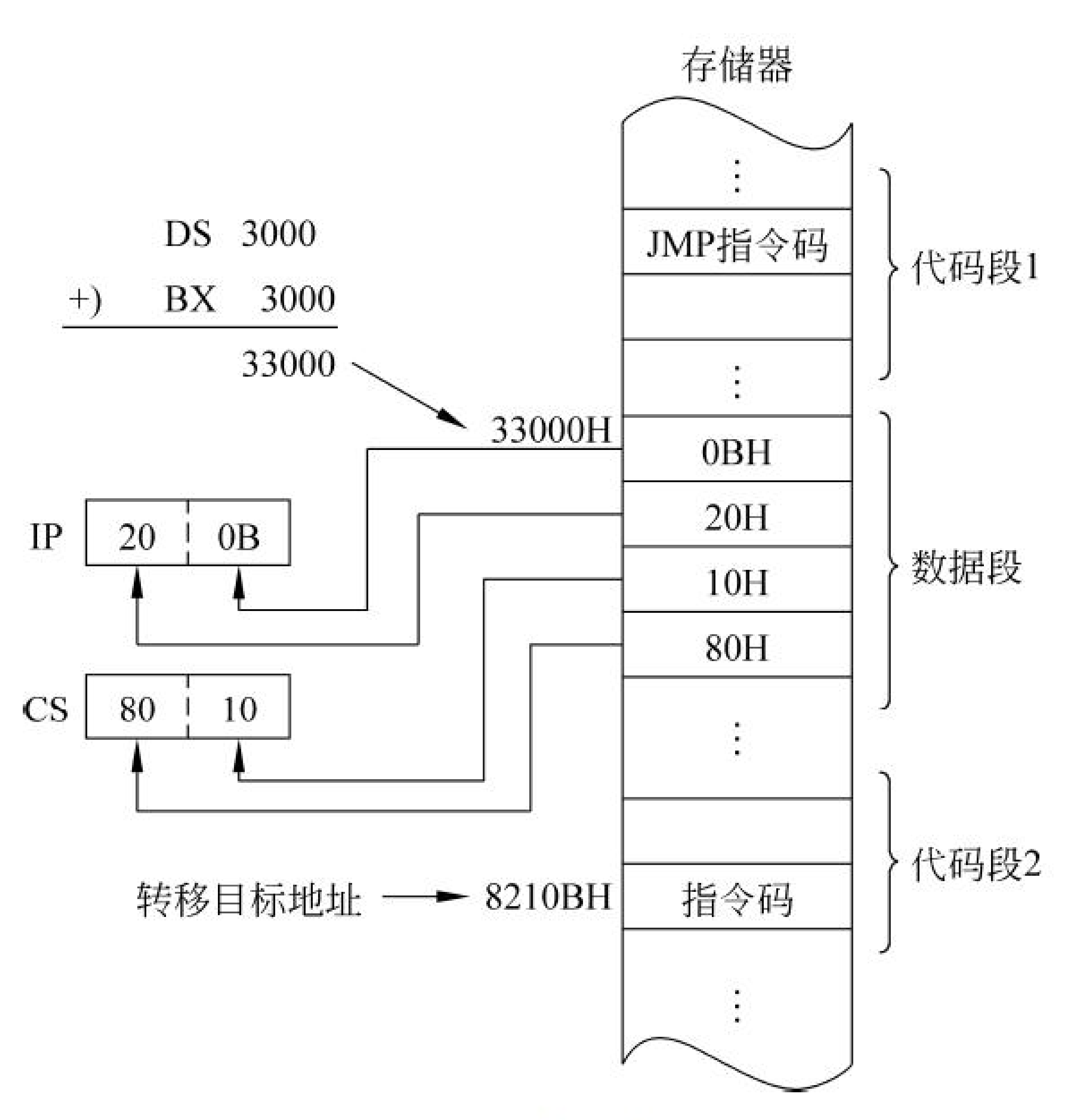


图 3-22 段间间接转移操作示意图

表 3-4 条件转移指令

指令名称	汇编格式	转移条件	备注
CX 内容为 0 转移	JCXZ target	CX=0	
大于/不小于等于转移	JG/JNLE target	SF=OF 且 ZF=0	带符号数
大于等于/不小于转移	JGE/JNL target	SF=OF	带符号数
小于/不大于等于转移	JL/JNGE target	SF≠OF 且 ZF=0	带符号数
小于等于/不大于转移	JLE/JNG target	SF≠OF 或 ZF=1	带符号数
溢出转移	JO target	OF=1	
不溢出转移	JNO target	OF=0	
结果为负转移	JS target	SF=1	
结果为正转移	JNS target	SF=0	
高于/不低于等于转移	JA/JNBE target	CF=0 且 ZF=0	无符号数
高于等于/不低于转移	JAE/JNB target	CF=0	无符号数
低于/不高于等于转移	JB/JNAE target	CF=1	无符号数
低于等于/不高于转移	JBE/JNA target	CF=1 或 ZF=1	无符号数
进位转移	JC target	CF=1	
无进位转移	JNC target	CF=0	
等于或为零转移	JE/JZ target	ZF=1	
不等于或非零转移	JNE/JNZ target	ZF=0	
奇偶校验为偶转移	JP/JPE target	PF=1	
奇偶校验为奇转移	JNP/JPO target	PF=0	

由于条件转移指令是根据状态标志位的状态决定是否转移的，因此在使用时，其前一条指令应是执行后能够对相应状态标志位产生影响的指令。例如，要判断两个无符号数

的大小，应当用 CMP 指令，然后根据执行后 CF 的状态，在其后使用 JNC（或 JC）指令决定如果目标操作数大（或小）程序转移到何处执行。

在有些情况下，需要用两个或两个以上标志位的状态组合来判断是否实现转移。例如，对有符号数的比较需根据符号标志 SF 和溢出标志 OF 的组合来判断，若包含“等于”条件，还需组合 ZF 标志。

【例 3-43】 在内存的数据段中存放了 100 个 8 位带符号数，其首地址为 TABLE，试统计其中正元素、负元素和零元素的个数，并分别将个数存入 PLUS、MINUS 和 ZERO 这 3 个单元中。

题目分析：

为实现上述功能，可先将 PLUS、MINUS 和 ZERO 这 3 个单元清零，之后将数据表中的带符号数逐个放入 AL 中，再利用条件转移指令测试该数是正数、负数还是零，再分别在对应的单元中计数。程序如下：

```
START: XOR AL,AL                ;AL清零
       MOV PLUS,AL              ;PLUS单元清零
       MOV MINUS,AL             ;MINUS单元清零
       MOV ZERO,AL              ;ZERO单元清零
       LEA SI,TABLE             ;数据表首地址送 SI
       MOV CL,100               ;表长度送 CL
       CLD                      ;使 DF= 0
CHECK: LODSB                    ;取一个数到 AL
       OR AL,AL                 ;操作数自身相"或",仅影响标志位
       JS X1                    ;若为负,转 X1
       JZ X2                    ;若为零,转 X2
       INC PLUS                 ;否则为正,PLUS单元加 1
       JMP NEXT                 ;
   X1: INC MINUS                ;MINUS单元加 1
       JMP NEXT                 ;
   X2: INC ZERO                 ;ZERO单元加 1
NEXT:  DEC CL                   ;CL减 1
       JNZ CHECK                ;若 ZF= 0转 CHECK
       HLT                      ;停止
```

3. 循环控制指令

循环控制指令顾名思义是在循环程序中用来控制循环的。其控制转向的目标地址是以当前 IP 内容为中心的 −128～＋127 字节范围内。循环次数必须预先送入 CX 寄存器中。一般情况下，循环控制指令放在循环程序的开始或结尾。

循环控制指令共有 3 条，它们均不影响标志位。

1）LOOP 指令

指令格式：

```
LOOP LABEL
```

这里的 LABEL 相当于一个近地址标号。指令的执行是先将 CX 内容减 1,再判断 CX 是否为 0,若 CX≠0,则转至目标地址继续循环;否则就退出循环,执行下一条指令,即,LOOP 指令相当于以下两指令的组合。

```
DEC CX
JNZ NEXT
```

【例 3-44】 在以 DATA 为首地址的内存数据段中存放有 200 个 16 位带符号数,试找出其中最大和最小的符号数,并分别放在以 MAX 和 MIN 为首的内存单元中。

题目分析:

为寻找最大和最小的数,可先取出数据段中的一个数据作为标准,将其同时暂存于 MAX 和 MIN 中,然后使其他数据分别与 MAX 和 MIN 中的数进行比较,若大于 MAX 内容,则取代原 MAX 中的数;若小于 MIN 内容,则将新数放于 MIN 中,最后就得出了数据段中最大和最小的带符号数。要注意的一点是:比较带符号数的大小时应采用 JG 和 JL 等用于符号数的条件转移指令。程序如下:

```
 START: LEA SI,DATA          ;SI←数据段首地址
        MOV CX,200           ;CX←数据段长度
        CLD                  ;清方向标志 DF
        LODSW                ;AX←一个 16位带符号数
        MOV MAX,AX           ;将该数送 MAX
        MOV MIN,AX           ;将该数送 MIN
        DEC CX               ;CX←CX-1
  NEXT: LODSW                ;取下一个 16位带符号数
        CMP AX,MAX           ;与 MAX单元内容进行比较
        JG LARGER            ;若大于则转 LARGER
        CMP AX,MIN           ;否则再与 MIN单元内容进行比较
        JL SMALL             ;若小于 MIN的内容则转 SMALL
        JMP GOON             ;否则就转至 GOON
LARGER: MOV MAX,AX           ;MAX←AX
        JMP GOON             ;
 SMALL: MOV MIN,AX           ;MIN←AX
  GOON: LOOP NEXT            ;CX-1,若 CX≠0则转 NEXT
        HLT
```

2) LOOPZ(或 LOOPE)指令

指令格式:

```
LOOPZ LABEL
```

或

```
LOOPE LABEL
```

LOOPZ 指令在执行时先使 CX 内容减 1，再根据 CX 中的值及 ZF 的值来决定是否继续循环。继续循环的条件是 CX≠0，且 ZF=1；若 CX=0 或者 ZF=0，则退出循环。

3）LOOPNZ(或 LOOPNE)指令

指令格式：

```
LOOPNZ LABEL
```

或

```
LOOPNE LABEL
```

LOOPNZ 指令与 LOOPZ 指令类似，只是其中 ZF 条件与之相反。它先将 CX 内容减 1，然后再判断 CX 和 ZF 的内容，当 CX≠0 且 ZF=0 的条件下，就转至目标地址继续循环；否则退出循环。

【例 3-45】 比较两组输入端口的数据是否一致。主端口的首地址为 MAIN_PORT，冗余端口的首地址为 REDUNDANT_PORT，端口数目均为 NUMBER。

```
            MOV DX,MAIN_PORT          ;DX←主端口地址指针
            MOV BX,REDUNDANT_PORT     ;BX←冗余端口地址指针
            MOV CX,NUMBER             ;CX←端口数
      TOP:  IN AX,DX                  ;AX←从主端口输入一个数据
            XCHG AX,BP                ;主端口输入的数据暂存于 BP
            INC DX                    ;主端口地址指针加 1
            XCHG BX,DX                ;DX←冗余端口地址指针
            IN AX,DX                  ;AX←从冗余端口输入一个数据
            INC DX                    ;冗余端口地址指针加 1
            XCHG BX,DX                ;两端口地址指针恢复到原寄存器中
            CMP AX,BP                 ;比较两端口的数据
            LOOPE TOP                 ;若两端口数据相等且 CX-1≠0,则转 TOP
            JNZ PORT_ERROR            ;若两端口数据不相等,则转至 PORT_ERROR
            ⋮
PORT_ERROR: …
            ⋮
```

4. 过程调用和返回

在编程过程中，为了节省内存单元，往往将程序中常用到的具有相同功能的部分独立出来，编写成一个模块，称之为子程序（或过程）。程序执行中，主程序在需要时可随时调用这些子程序；子程序执行完以后，又返回到主程序继续执行。在需要时还可多级调用，如图 3-23 所示。8086/8088 指令系统为实现这一功能提供了调用指令 CALL 和返回指令 RET。

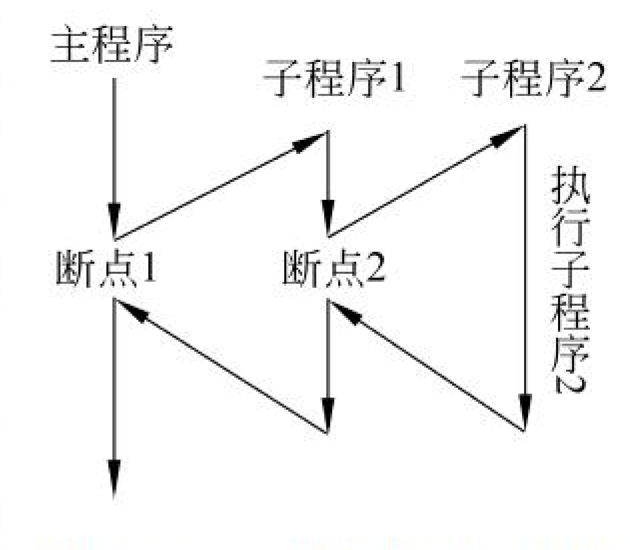


图 3-23　子程序调用示意图

调用指令 CALL 执行时，CPU 先将下一条指令的地址（称为返回地址）压入堆栈保护起来，然后将子程序入口地址赋给 IP（或 CS 和 IP）中，以便转到子程序执行。

返回指令 RET 一般安排在子程序末尾，执行 RET 时，CPU 将堆栈顶部保留的返回地址弹出到 IP(或 CS 和 IP)中，这样即可返回到 CALL 的下一条指令，继续执行主程序。

由于子程序有可能与主程序同在一个段内，也有可能不同在一个段内。所以与无条件转移指令一样，CALL 指令也有 4 种形式，即段内直接调用、段内间接调用、段间直接调用以及段间间接调用。

1）段内直接调用

指令格式：

```
CALL NEAR PROC
```

这里，PROC 是一个近过程的符号地址，表示指令调用的过程是在当前代码段内。指令在汇编后会得到 CALL 指令的下一条指令与被调用过程的入口地址之间相差的 16 位相对位移量(也可以理解为是字节表示的距离)。

CALL 指令执行时，首先将下面一条指令的偏移地址压入堆栈，然后将指令中 16 位的相对位移量和当前 IP 的内容相加，新的 IP 内容即为所调用过程的入口地址(确切地说是入口地址的偏移地址)。执行过程表示如下。

$SP \leftarrow SP-2$

$SP+1 \leftarrow IP_H$

$SP \leftarrow IP_L$

$IP \leftarrow IP+16$位偏移量

对于段内调用，指令中的 NEAR 可以省略。例如“CALL TIME”指令将调用一个名为 TIME 的近过程。

2）段内间接调用

指令格式：

```
CALL OPRD
```

这里，OPRD 为 16 位寄存器或两个存储器单元的内容。这个内容代表的是一个近过程的入口地址。指令的操作是将 CALL 指令的下面一条指令的偏移地址压入堆栈，若指令中的操作数(OPRD)是一个 16 位通用寄存器，则将寄存器的内容送 IP；若是存储单元，则将存储器的两个单元的内容送 IP。例如：

```
CALL AX                 ;IP ← AX,子程序的入口地址由 AX 给出
CALL WORD PTR[BX]       ;IP← ([BX+ 1]:[BX])
                        ;子程序的入口地址为数据段 [BX]和 [BX+ 1]两存储单元的内容
```

3）段间直接调用

指令格式：

```
CALL FAR PROC
```

这里，PROC 是一个远过程的符号地址，表示指令调用的过程在另外的代码段内。

指令在执行时先将 CALL 指令的下一条指令的地址，即 CS 和 IP 寄存器的内容压入堆栈，然后用指令中给出的段地址取代 CS 的内容，偏移地址取代 IP 的内容。执行过程如下。

```
SP←SP-2,([SP+1]:[SP])←CS          ;CS←被调用过程入口的段地址
SP←SP-2,([SP+1]:[SP])←IP          ;IP←被调用过程入口的偏移地址
```

例如，指令“CALL 3000H:2100H”直接给出了被调用过程的段地址和偏移地址“3000H:2100H”。

4）段间间接调用

指令格式：

```
CALL OPRD
```

这里，OPRD 为 32 位的存储器地址。指令的操作是将 CALL 指令的下一条指令的地址，即 CS 和 IP 的内容压入堆栈，然后把指令中指定的连续 4 个存储单元中的内容送 IP 及 CS，低地址的两个单元内容为偏移地址，送入 IP；高地址的两个单元内容为段地址，送入 CS。

【例 3-46】 设 DS＝6000H，SI＝0560H，执行指令 CALL DWORD PTR[SI]。

该指令表示所调用程序的入口地址存放在当前数据段中 SI 的内容为首地址的连续 4 个字节单元中。指令操作示意图如图 3-24 所示。

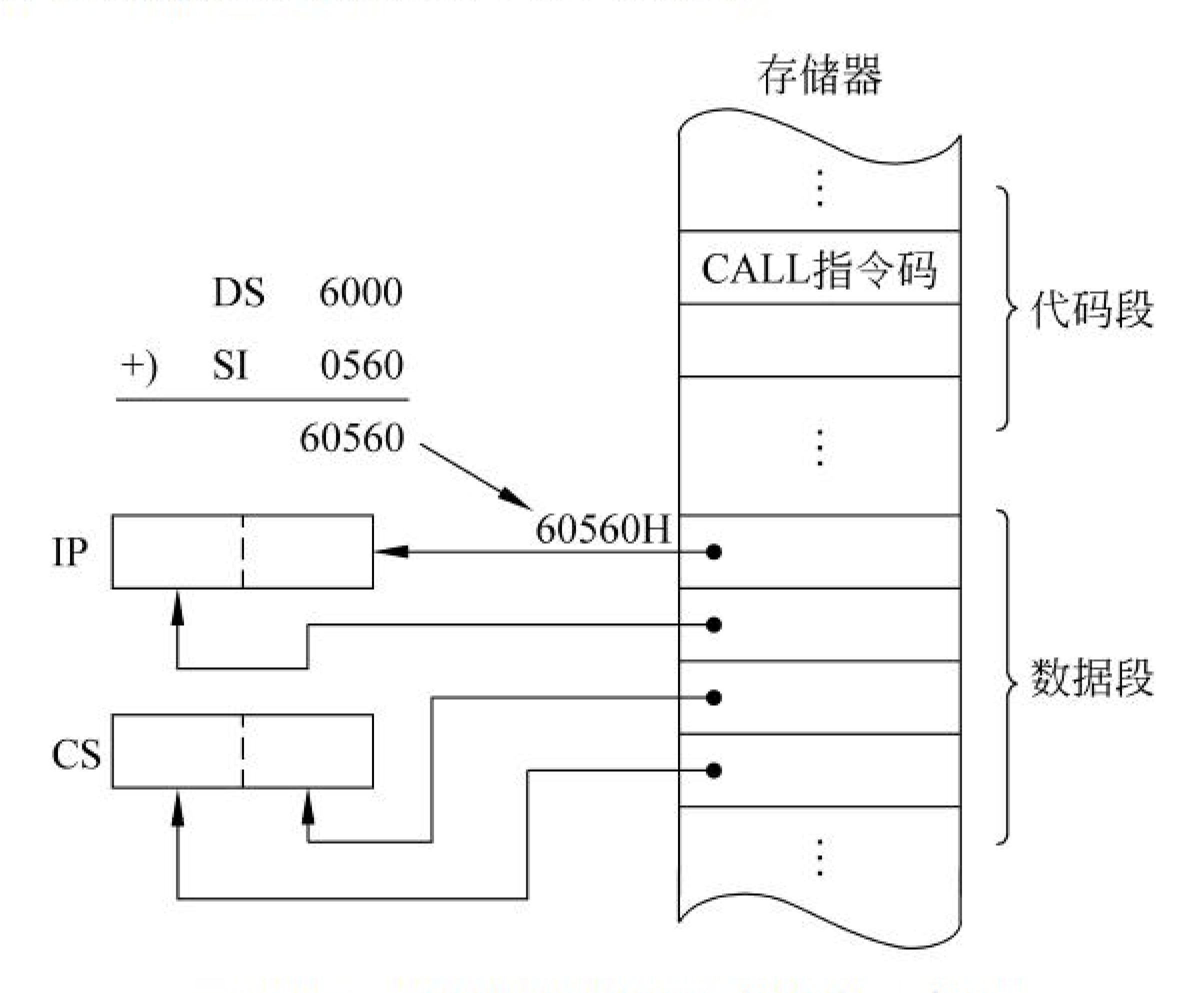


图 3-24 段间间接调用指令操作示意图

5）返回指令 RET

指令格式：

```
RET
```

返回指令执行与调用指令相反的操作。对于近过程(与主程序在同一段内)，用 RET 返回主程序时，只需从堆栈顶部弹出一个字的内容给 IP 作为返回的偏移地址；对于远过程(与主程序不在同一段)，用 RET 返回主程序时，则需从堆栈顶部弹出两个字作为返回

地址,先弹出一个字的内容给 IP 作为返回的偏移地址,再弹出一个字的内容给 CS 作为返回的段地址。

无论是段间返回还是段内返回,返回指令在形式上都是 RET。

返回指令一般作为子程序的最后一条语句。所有的返回指令都不影响标志位。

5. 中断指令

所谓中断,是指在程序运行期间因某种随机或异常的事件,要求 CPU 暂时中止正在运行的程序转去执行一组专门的中断服务程序来处理这些事件,处理完毕后又返回到原被中止处继续执行原程序的过程。

引起中断的事件叫作中断源,它可以是在 CPU 内部,也可以是在 CPU 外部。内部中断源引起的中断称为内部中断;相应地,外部中断源引起的中断就称为外部中断。8086/8088 中断系统分为外部中断(或叫硬件中断)和内部中断(或叫软件中断)。外部中断主要用来处理外设和 CPU 之间的通信。内部中断包括运算异常及中断指令引起的中断。

中断指令用于产生软件中断,以执行一段特殊的中断处理过程。中断指令主要有以下几个用途。

(1) 用户程序可通过中断指令调用操作系统提供的特殊子程序(称为系统功能调用)。这些特殊子程序为用户程序提供了控制台输入输出、文件系统、软硬件资源管理、通信等丰富的服务。在用户程序中只要用一条中断指令即可使用这些服务,而不用再自己编写类似的程序,大大简化了应用软件的开发。

(2) 用来实现一些特殊的功能,如调试程序时单步运行、断点等。

(3) 调用 BIOS 提供的硬件低层服务。

关于中断,将在本书第 7 章进行详细介绍,这里仅介绍指令的格式及操作。8086/8088 指令系统提供了 3 条与软件中断相关的指令。

1) INT 指令

指令格式:

```
INT n
```

这里,n 为中断向量码(也称中断类型码),是一个常数,取值范围为 0～255。

指令执行时,CPU 根据 n 的值计算出中断向量的地址,然后从该地址中取出中断服务程序的入口地址,并转到该中断服务子程序去执行。中断向量地址的计算方法是将中断向量码 n 乘 4。INT 指令的具体操作步骤如下。

(1) SP←SP-2,([SP+1]:[SP])←FLAGS ;把标志寄存器的内容压入堆栈

(2) TF←0,IF←0

;清除 IF 和 TF,保证不会中断正在执行的中断子程序,并且不响应单步中断

(3) SP←SP－2,([SP+1]:[SP])←CS

SP←SP－2,([SP+1]:[SP])←IP

;把断点地址(即 INT 指令的下一条指令的地址)压入堆栈

(4) IP←([n×4+1]:[n×4]) CS←([n×4+3]:[n×4+2])

由 n×4 得到中断向量地址,进而得到中断处理子程序的入口地址。

以上操作完成后,CS:IP 就指向中断服务程序的第一条指令,此后 CPU 开始执行中断服务子程序。

INT n 指令除复位 IF 和 TF 外,对其他标志无影响。

从 CPU 执行中断指令的过程可以看出,INT 指令的基本操作与存储器寻址的段间间接调用指令非常相像,所不同的有以下 3 点。

(1) INT 指令要把标志寄存器 FLAGS 压入堆栈,而 CALL 指令不保存 FLAGS 内容。

(2) INT 影响 IF 和 TF 标志,而 CALL 指令不影响。

(3) 中断服务程序入口地址放在内存的固定位置,以便通过中断向量码找到它,而 CALL 指令可任意指定子程序入口地址的存放位置。

2) 中断返回指令 IRET

中断返回指令 IRET 用于从中断服务子程序返回到被中断的程序继续执行。任何中断服务子程序无论是由外部中断引起的,还是内部中断引起的,其最后一条指令都是 IRET 指令。该指令首先将堆栈中的断点地址弹出到 IP 和 CS,接着将 INT 指令执行时压入堆栈的标志字弹出到标志寄存器以恢复中断前的标志状态。显然本指令对各标志位均有影响。指令的操作为

(1) IP←([SP+1]:[SP]),SP←SP+2

(2) CS←([SP+1]:[SP]),SP←SP+2

(3) FLAGS←([SP+1]:[SP]),SP←SP+2

3.3.6 处理器控制指令

处理器控制指令用来对 CPU 进行控制,如修改标志寄存器、使 CPU 暂停、使 CPU 与外部设备同步等,共分为两大类:标志操作指令和外部同步指令。各指令的功能如表 3-5 所示。

表 3-5 处理器控制指令

汇编格式		操作	
标志位操作指令	CLC	CF←0	;清进位标志位
	STC	CF←1	;进位标志位置位
	CMC	CF←$\overline{CF}$	;进位标志位取反
	CLD	DF←0	;清方向标志位,串操作从低地址到高地址
	STD	DF←1	;方向标志位置位,串操作从高地址到低地址
	CLI	IF←0	;清中断标志位,即关中断
	STI	IF←1	;中断标志位置位,即开中断

续表

汇编格式		操作
外部同步指令	HLT	暂停指令，使 CPU 处于暂停状态，常用于等待中断的产生
	WAIT	当$\overline{\text{TEST}}$引脚为高电平（无效）时，执行 WAIT 指令会使 CPU 进入等待状态；主要用于 8088 与协处理器和外部设备的同步
	ESC	处理器交权指令，用于与协处理器配合工作时
	LOCK	总线锁定指令，主要为多机共享资源设计
	NOP	空操作指令，消耗 3 个时钟周期，常用于程序的延时等

3.4 Pentium 新增指令简介

8086/8088 指令系统是 80x86 系列 CPU 的基本指令系统，它的指令编码、寻址方式与 Intel 80x86 系列 CPU 运行在实地址模式下是完全相同的。由于从 80386 起增加了虚地址模式，因此相应地增加了虚地址模式下的寻址方式，其指令系统也随之扩充，功能进一步增强。本节以 80386 为例，简要介绍 80x86 CPU 在 8086 指令系统基础上新增的指令功能及寻址方式。

3.4.1 80x86 虚地址下的寻址方式

相对于实地址模式下的 8 种对操作数的寻址方式，80x86 增加了虚地址下寻址 32 位数的寻址能力，如表 3-6 所示。

表 3-6 80x86 寻址方式

寻址方式	操作示例	
立即寻址	MOV EAX，12345678H	；将 32 位立即数送 32 位寄存器 EAX
直接寻址	MOV EAX，[11202020H]	；直接给出 32 位地址
寄存器寻址	MOV EAX，EBX	；将 32 位寄存器 EBX 的内容送到 EAX
寄存器间接寻址	MOV EBX，[EAX]	；将数据段中偏移地址为 EAX 内容的 ；4 个字节数送 EBX
寄存器相对寻址	MOV AX，DATA[EBX]	；将 EBX 的内容与 32 位位移量 DATA ；的和所指的两单元的内容送 AX
基址、变址寻址	MOV EAX，[EBX][ESI]	；将数据段中 EBX＋ESI 所指的 4 单元 ；内容送 EAX
基址、变址、相对寻址	MOV EAX，[EBX＋EDI＋0FFFFFF0H]	

续表

寻址方式	操作示例	
带比例因子的变址寻址	MOV EAX,DATA[ESI×4]	;将变址寄存器 ESI 的内容乘上一个 ;比例因子,再加上位移量形成存放 ;操作数的有效偏移地址
带比例因子的基址、变址寻址	MOV EBX,[EDX×4][EAX]	;将数据段中(EDX×4)+EAX 所指 4 ;单元的内容送 EBX
带比例因子的基址、变址、相对寻址	MOV EAX,[EBX+DATA][EDI×4]	

注:

(1) 80x86 允许所有的通用寄存器都可用作间接寻址,除 ESP 和 EBP 默认数据在堆栈段外,其他通用寄存器作间址寄存器时,都默认数据在数据段,但允许段重设。

(2) 在基址、变址、相对寻址方式中,当位移量是 32 位时,基址寄存器和变址寄存器可以是任何一个通用寄存器,由基址寄存器决定数据默认在哪一个段。

(3) 在带比例因子的变址寻址中,比例因子的选取与操作数的字长相同,如操作数可以是 1 字节、2 字节、4 字节或 8 字节,相应地,比例因子可以是 1、2、4 或 8,乘比例因子的那个寄存器被认为是变址寄存器,操作数默认的段由选用的基址寄存器决定。

3.4.2 80x86 CPU 新增指令简述

随着 Intel 公司系列微处理器技术的发展,CPU 的字长由 16 位扩展到了 32 位,其指令系统也随之得到了相应的扩充和增强。除增强了部分 8086 指令的功能外,还增加了一些新的指令,以使程序的编写更加方便,并使整个系统的功能增强、执行速度提高。

从 80386 CPU 之后都是 32 位的微处理器,具有 32 位的内部通用寄存器和 32 位数据总线,可以进行 32 位数据的并行操作。它们的指令系统中除加强了 8086 部分指令的功能外,主要是增加了对 32 位数的操作,表 3-7 列出了部分主要新增指令及其功能。

表 3-7 80386 及以上微处理器主要新增或增强指令

指令类型	汇编格式	操作说明
数据传送及扩展指令	MOVSX reg,reg/mem	源操作数是 8 位/16 位寄存器/存储器,目标操作数是 16 位或 32 位的寄存器。指令的功能是将源操作数的符号位扩展后送到目标地址。若源操作数是 8 位,则扩展为 16 位;源操作数是 16 位,则扩展为 32 位
	MOVSX reg,reg/mem	与 MOVSX 的格式和操作相同,只是将高位全部扩展为 0
堆栈操作指令	PUSH/POP imm	imm 可以是 16 位或 32 位立即数
	PUSHA/POPA	保存/弹出全部 16 位寄存器集
	PUSHAD/POPAD	保存/弹出全部 32 位寄存器集
	PUSHFD/POPFD	保存/弹出 32 位标志寄存器
串输入串输出指令	INS(INSB/INSW/INSD 等)	从 I/O 设备传送字节、字或双字数据到 DI 寻址的附加段内的存储单元
	OUTS(OUTSB/OUTSW/OUTSD 等)	从 SI 寻址的数据段存储单元把字节、字或双字数据传送到 I/O 设备

续表

指令类型	汇编格式	操 作 说 明
* 字节交换指令	BSWAP reg	将给定 32 位寄存器内的第 1 字节与第 4 字节及第 2 字节与第 3 字节交换
** 条件传送指令	CMOV(CMOVB/CMOVS/CMOVZ 等)	指令根据当前标志位的状态，决定是否进行数据传送
* 交换并相加指令	XADD reg/mem，reg	指令中的操作数可以是 8 位、16 位或 32 位的寄存器或存储器，指令的执行将目标操作数和源操作数相加，结果送回目标地址；同时，目标地址中的原值送入源操作数地址中。同加法指令一样，指令的执行会对状态标志位产生影响
* 比较交换指令	CMPXCHG reg/mem，reg	使目标操作数与累加器内容比较，若相等，将源操作数复制到目标操作数；若不相等，就将目标操作数复制到累加器中
双精度移位指令	SHRD reg/mem，reg，imm	将目标操作数中的内容逻辑右移 imm 指定的位数，移位后，中间操作数中右边的 imm 位移入目标操作数左边 imm 位中
	SHLD reg/mem，reg，imm	将目标操作数中的内容逻辑左移 imm 指定的位数，移位后，中间操作数中最左边的 imm 位移入目标操作数最右边 imm 位中
位测试与置位指令	BT reg/mem，reg BT reg/mem，imm	测试目标操作数中由源操作数所指定的位的状态，并将该位的状态复制到进位标志位 CF 中
	BTC reg/mem，reg BTC reg/mem，imm	测试目标操作数中由源操作数所指定的位的状态，并将该位取反后复制到 CF 中
	BTR reg/mem，reg BTR reg/mem，imm	测试目标操作数中由源操作数所指定的位的状态，并在将该位复制到 CF 中后将该位清“0”
	BTS reg/mem，reg BTS reg/mem，imm	测试目标操作数中由源操作数所指定的位的状态，并在将该位复制到 CF 中后将该位置“1”
高级语言类	BOUND reg，mem（数组边界检查指令）	源操作数是两个存储单元，其内容分别表示上界和下界。指令用于测试目标寄存器中的内容是否属于上下界之内，若不属于则产生 5 号中断，否则不做任何操作
	ENTER OPRD1，OPRD2（设置堆栈空间指令）	为高级语言正在执行的过程设置堆栈空间。OPRD1 是 16 位常数，表示堆栈区域的字节数；OPRD2 是 8 位常数，表示允许过程嵌套的层数
	LEAVE（撤销堆栈空间指令）	撤销 ENTER 指令所设置的堆栈空间。一般与 ENTER 指令配对使用
控制保护类	LAR（装入访问权限）	LSL（装入段限符）
	LGDT（装入全局描述符表）	SGDT（存储全局描述符表）
	LIDT（装入 8 字节中断描述符表）	SIDT（存储 8 字节中断描述符表）
	LLDT（装入局部描述符表）	SLDT（存储局部描述符表）
	LTR（装入任务寄存器）	STR（存储任务寄存器）
	LMSW（装入机器状态字）	SMSW（存储机器状态字）
	VERR（存储器或寄存器读校验）	VERW（存储器或寄存器写校验）
	ARPL（调整已请求特权级别）	CLTS（清除任务转移标志）

注：reg 表示寄存器操作数；mem 表示存储器操作数；imm 表示立即数；* 表示仅在 Intel 80486 及其以上微处理器中使用；** 表示仅在 Intel Pentium 及其以上微处理器中使用。

习　题

3.1　什么叫寻址方式？8086/8088 CPU 共有哪几种寻址方式？

3.2　设 DS＝6000H，ES＝2000H，SS＝1500H，SI＝00A0H，BX＝0800H，BP＝1200H，字符常数 VAR 为 0050H。请分别指出下列各条指令源操作数的寻址方式，并计算除立即寻址外的其他寻址方式下源操作数的物理地址。

(1) MOV AX,BX　　(2) MOV DL,80H
(3) MOV AX,VAR　　(4) MOV AX,VAR[BX][SI]
(5) MOV AL,'B'　　(6) MOV DI,ES:[BX]
(7) MOV DX,[BP]　　(8) MOV BX,20H[BX]

3.3　假设 DS＝212AH，CS＝0200H，IP＝1200H，BX＝0500H，位移量 DATA＝40H，[217A0H]＝2300H，[217E0H]＝0400H，[217E2H]＝9000H。试确定下列转移指令的转移地址。

(1) JMP BX
(2) JMP WORD PTR[BX]
(3) JMP DWORD PTR[BX+DATA]

3.4　试说明指令 MOV BX,5[BX]与指令 LEA BX,5[BX]的区别。

3.5　设堆栈指针 SP 的初值为 2300H，AX＝50ABH，BX＝1234H。执行指令 PUSH AX 后，SP＝？再执行指令 PUSH BX 及 POP AX 之后，SP＝？AX＝？BX＝？

3.6　判断下列指令是否正确，若有错误，请指出并改正之。

(1) MOV AH,CX　　(2) MOV 33H,AL
(3) MOV AX,[SI][DI]　　(4) MOV [BX],[SI]
(5) ADD BYTE PTR[BP],256　　(6) MOV DATA[SI],ES:AX
(7) JMP BYTE PTR[BX]　　(8) OUT 230H,AX
(9) MOV DS,BP　　(10) MUL 39H

3.7　已知 AL＝7BH，BL＝38H，试问执行指令 ADD AL,BL 后，AF、CF、OF、PF、SF 和 ZF 的值各为多少？

3.8　试比较无条件转移指令、条件转移指令、调用指令和中断指令的异同。

3.9　试判断下列程序执行后 BX 中的内容。

```
MOV CL,3
MOV BX,0B7H
ROL BX,1
ROR BX,CL
```

3.10　按下列要求写出相应的指令或程序段。

(1) 写出两条使 AX 内容为 0 的指令。
(2) 使 BL 寄存器中的高 4 位和低 4 位互换。

(3) 屏蔽 CX 寄存器的 D_{11}、D_7 和 D_3 位。

(4) 测试 DX 中的 D_0 和 D_8 位是否为 1。

3.11 分别指出以下两个程序段的功能。

(1)
```
MOV CX,10
LEA SI,FIRST
LEA DI,SECOND
STD
REP MOVSB
```

(2)
```
CLD
LEA DI,[1200H]
MOV CX,0F00H
XOR AX,AX
REP STOSW
```

3.12 执行以下两条指令后,标志寄存器 FLAGS 的 6 个状态位各为什么状态?

```
MOV AX,84A0H
ADD AX,9460H
```

3.13 将+46 和-38 分别乘以 2,可应用什么指令来完成?如果除以 2 呢?

3.14 已知 AX=8060H,DX=03F8H,端口 PORT1 的地址是 48H,内容为 40H;PORT2 的地址是 84H,内容为 85H。指出下列指令执行后的结果。

(1) OUT DX,AL

(2) IN AL,PORT1

(3) OUT DX,AX

(4) IN AX,48H

(5) OUT PORT2,AX

3.15 试编写程序,统计 BUFFER 为起始地址的连续 200 个单元中 0 的个数。

3.16 写出完成下述功能的程序段。

(1) 从地址 DS:0012H 中传送一个数据 56H 到 AL 寄存器。

(2) 将 AL 中的内容左移两位。

(3) AL 的内容与字节单元 DS:0013H 中的内容相乘。

(4) 乘积存入字单元 DS:0014H 中。

3.17 若 AL=96H,BL=12H,在分别执行指令 MUL 和 IMUL 后,其结果是多少?OF=?CF=?

第4章 汇编语言程序设计

引言：

家庭安全防盗系统需要软件的支持，虽然已有多种更接近于人类自然语言的高级语言问世，但汇编语言以其执行速度快和能够实现对硬件的直接控制等独特优点，依然应用于实时控制系统、嵌入式系统等软件开发的应用中。由于它是底层语言，因此学习汇编程序，特别有助于对计算机基本工作过程的理解。本章介绍汇编语言源程序的基本结构、汇编语言的语法及程序设计的基本方法。通过这一章的学习，应能掌握基本的汇编语言程序设计技术。

教学目的：

(1) 了解汇编语言源程序的结构；
(2) 深入理解伪指令系统；
(3) 深入理解 DOS 功能调用；
(4) 掌握汇编语言源程序的设计方法。

4.1 汇编语言源程序

任何一段计算机程序都是用某种计算机语言来编写的。根据计算机语言是更接近人类还是更接近于计算机，可将其分成高级语言和低级语言。低级语言包括机器语言和汇编语言两种。

机器语言(Machine Language)是用二进制码来表示指令和数据的语言，是计算机硬件系统唯一能够直接理解和执行的语言，具有执行速度快、占用内存少等优点。但是其不直观、不易理解和记忆，因此编写、阅读和修改程序都比较麻烦。

汇编语言(Assembly Language)弥补了机器语言的不足，它用指令助记符、符号地址、标号和伪指令等来书写程序。由于助记符接近于自然语言，因此与机器语言相比，它在程序的编写、阅读和修改方面都比较方便、不易出错，且执行速度和机器语言程序相同。

用汇编语言编写的程序称为汇编语言源程序。由于计算机只能辨认和执行机器语言，因此必须将汇编语言源程序“翻译”成能够在计算机上执行的机器语言(称为目标代码程序)，这个翻译的过程称为汇编(Assemble)，完成汇编过程的系统程序叫做汇编程序(Assembler)。目前使用较多的汇编程序称为宏汇编(MASM)程序。它除了能将源程序

翻译成目标代码外，还提供了很多增强功能，如允许使用宏定义以简化编程；能检查出源程序编写过程中出现的语法错误；还可根据用户要求自动分配各类存储区（程序区、数据区等）；能自动将非二进制数转换为二进制数，自动进行字符到 ASCII 码的转换以及计算指令中表达式值；等等。

汇编语言和机器语言一样，都是面向具体机器的语言。也就是说，不同种类的 CPU 具有不同的汇编语言，互相之间不能通用（但同一系列的 CPU 是向前兼容的）。例如，x86 系列 CPU（包括 Intel 公司的 8088/8086/…/Pentium 和 AMD 公司的 K5/K6/K7 等）的汇编语言程序就不能在 PowerPC 系列的 CPU 上运行。这是它与高级语言很本质的区别之一。因此，使用汇编语言编写程序需要对它所适用的计算机系统的结构及工作原理有一定的了解。

与上述两种语言相比，高级语言（High Level Language）的语句更接近人类语言，所以用高级语言编写的程序易读、易编，相对比较简短。它与具体的计算机无关，不受 CPU 类型的限制，通用性很强。用高级语言编程不需了解计算机内部的结构和原理，对于非计算机专业的人员来讲比较易于掌握。用高级语言编写的源程序同样必须"翻译"成为机器代码计算机才能执行，完成这个"翻译"过程的系统软件称为编译程序或解释程序。它通常要比汇编程序复杂得多，需要占用更多的内存，编译或解释的过程也要花费更多的时间。

目前，随着计算机技术的发展，人们已极少再直接使用机器语言编写程序。汇编语言主要应用在对程序执行速度要求较高而内存容量又有限的场合（如某些工控和实时控制系统中）或需要直接访问硬件的场合等。高级语言的优势是众所周知的，但它也有需要内存容量大、执行速度相对较慢等缺点。为了扬长避短，有时在一个程序中对执行速度或实时性要求较高的部分用汇编语言编写，而其余部分则可用高级语言编写。

4.1.1 汇编语言源程序的结构

在第 3 章关于指令系统的介绍中，曾列举过一些用汇编语言编写的程序。但是这些程序都不是完整的汇编语言源程序，在计算机上不能通过汇编生成目标代码，当然也就不能在机器上运行。那么，完整的汇编语言源程序是什么样的呢？

一个完整的汇编语言源程序通常由若干个逻辑段（Segment）组成，包括数据段、附加段、堆栈段和代码段，它们分别映射到存储器中的物理段上。每个逻辑段以 SEGMENT 语句开始，以 ENDS 语句结束，整个源程序用 END 语句结尾。

代码段中存放源程序的所有指令码，数据、变量等则放在数据段和附加段中。程序中可以定义堆栈段，也可以直接利用系统中的堆栈段。具体一个源程序中要定义多少个段应根据实际需要来定。但一般来说，一个源程序中可以有多个代码段，也可以有多个数据段、附加段及堆栈段，但一个源程序模块只可以有一个代码段、一个数据段、一个附加段和一个堆栈段。将源程序以分段形式组织是为了在程序汇编后，能将指令码和数据分别装入存储器的相应物理段中。

为了帮助读者建立起汇编语言源程序的整体结构，下面先给出一个完整的汇编语言源程序的结构框架，具体内容将在 4.2 节伪指令部分做详细介绍。

```
段名 1 SEGMENT
⋮
段名 1 ENDS
段名 2 SEGMENT
⋮
段名 2 ENDS
…
段名 n SEGMENT
⋮
段名 n ENDS
END
```

下面以一个具体的例子来说明一个完整汇编语言程序的结构。

【例 4-1】 编写一个两个字相加的程序。程序如下：

```
DSEG SEGMENT                    ;定义数据段
DATA1 DW 0F865H                 ;定义被加数
DATA2 DW 360CH                  ;定义加数
DSEG ENDS                       ;数据段结束
;
ESEG SEGMENT                    ;定义附加段
SUM DW 2 DUP(?)                 ;定义存放结果区
ESEG ENDS                       ;附加段结束
;
CSEG SEGMENT                    ;定义代码段
;下面的语句说明程序中定义的各段分别用哪个段寄存器寻址
ASSUME CS:CSEG,DS:DSEG,ES:ESEG
START: MOV AX,DSEG
       MOV DS,AX                ;初始化 DS
       MOV AX,ESEG
       MOV ES,AX                ;初始化 ES
       LEA SI,SUM               ;存放结果的偏移地址送 SI
       MOV AX,DATA1             ;取被加数
       ADD AX,DATA2             ;两数相加
       MOV ES:[SI],AX           ;和送附加段的 SUM 单元中
       HLT
CSEG ENDS                       ;代码段结束
     END START                  ;源程序结束
```

4.1.2 汇编语言语句类型及格式

汇编语言源程序的语句可分为两大类：指令性语句和指示性语句。

指令性语句是由指令助记符等组成的可被 CPU 执行的语句，第 4 章中介绍的所有指

令都属于指令性语句;指示性语句用于告诉汇编程序如何对程序进行汇编,是 CPU 不执行的指令,由于它并不能生成目标代码,故又称其为伪操作语句或伪指令。

汇编语言的语句由若干部分组成,指令性语句和指示性语句稍微有一点区别。

指令性语句的一般格式为

[标号:] [前缀] 操作码 [操作数[,操作数]] [;注释]

指示性语句的一般格式为

[名字] 伪操作 操作数 [,操作数,…]] [;注释]

其中,加方括号的是可选项,可以有,也可以没有,需要根据具体情况来定。

指令性语句和指示性语句在格式上的区别主要有以下两点。

(1)“标号”和“名字”:指令性语句中的“标号”和指示性语句中的“名字”在形式上类似,但标号表示指令的符号地址,需要加上“:”;名字通常表示变量名、段名和过程名等,其后不加“:”。不同的伪操作对于是否有名字有不同的规定,有些伪操作规定前面必须有名字,有些则不允许有名字,还有一些可以任选。名字在多数情况下表示的是变量名,用来表示存储器中一个数据区的地址。

(2) 指令性语句中的操作数最多为双操作数,也可以没有操作数,而指示性语句中的操作数至少要有一个,并可根据需要有多个,当操作数不止一个时,相互之间用逗号隔开。例如:

```
START:MOV AX,DATA              ;指令性语句,将立即数 DATA 送累加器 AX
DATA1 DB 11H,22H,33H           ;指示性语句,定义字节型数据。“DB”是伪操作
```

注释(Comment)是汇编语言语句的最后一个组成部分。它并不是必要的,加上的目的是增加源程序的可读性。对一个较长的应用程序来讲,如果从头到尾没有任何注释,读起来会很困难。因此,最好在重要的程序段前面以及关键的语句处加上简明扼要的注释。注释的前面要求加上分号“;”,注释可以跟在语句后面,也可作为一个独立的行。如果注释的内容较多,超过一行,则换行以后前面还要加上分号。注释不参加程序汇编,即不生成目标程序,它只是为程序员阅读程序提供方便。

指令性语句的操作码和前缀在第 3 章中已进行了详细的讨论,伪操作将在 4.2 节中介绍。下面主要讨论汇编语言语句中的操作数部分。

4.1.3 数据项及表达式

操作数是汇编语言语句中的一个重要组成部分。它可以是寄存器、存储器单元或数据项,而数据项又可以是常量、标号、变量和表达式。

1. 常量

常量(Constant)包括数字常量和字符串常量两种。数字常量可以用不同的数制表示。

(1) 十进制常量,以字母"D"(Decimal)结尾或不加结尾,如 23D,23。

(2) 二进制常量,以字母"B"(Binary) 结尾的二进制数,如 10101001B。

(3) 十六进制常量,以字母"H"(Hexadecimal)结尾,如 64H、0F800H。程序中,若是以字母 A～F 开始的十六进制数,在前面要加一个数字 0。

字符串常量是用单引号括起的一个或多个 ASCII 码字符。汇编程序将其中的每一个字符分别翻译成对应的一个字节的 ASCII 值,如'AB',汇编时将翻译为 41H、42H。

2. 标号

指令的标号(Label)是由程序员确定的,它不能与指令助记符或伪指令重名,也不允许由数字打头,字符个数不超过 31 个。

指令性语句中的标号代表存放一条指令的存储单元的符号地址,其后须加冒号。并不是每条指令性语句都必须有标号,但如果一条指令前面有一个标号,则程序中其他地方就可以引用这个标号,因此可以作为转移(无条件转移或条件转移)、过程调用以及循环控制等指令的操作数。

标号具有 3 种属性: 段值、偏移量和类型。

(1) 段值属性。段值属性是标号所在段的段地址,当程序中引用一个标号时,该标号应在代码段中。

(2) 偏移量属性。偏移量属性是标号所在段的段首到定义该标号的地址之间的字节数(即偏移地址)。偏移量是一个 16 位无符号数。

(3) 类型。标号的类型有两种: NEAR 和 FAR。前一种称为近标号,只能在段内被引用,地址指针为 2 个字节;后一种称为远标号,可以在其他段被引用,地址指针为 4 个字节。

3. 变量

变量(Variable)与标号一样也具有 3 种属性。变量的段属性就是它所在段的段地址,因为变量一般在存储器的数据段或附加段中,所以变量的段值在 DS 或 ES 寄存器中。

变量的偏移量属性是该变量所在段的起始地址到变量地址之间的字节数。

变量的类型有 BYTE(字节)、WORD(字)、DWORD(双字)、QWORD(四字)、TBYTE(十字节)等,表示数据区中存取操作对象的大小。

变量是存储器中某个数据区的名字,由于数据区中内容是可以改变的,因此变量的值也可以改变。变量在指令中可以作为存储器操作数引用。

变量名由字母开头,其长度不能超过 31 个字符,在使用变量时应注意以下两点。

(1) 变量类型与指令的要求必须相符。例如:

```
MOV AX,VAR1            ;要求 VAR1 必须定义为字类型变量,否则这里的引用就是错误的
MOV BL,VAR2            ;要求 VAR2 必须定义为字节型变量,否则这里的引用就是错误的
```

(2) 在定义变量时,变量名对应的是数据区的首地址。如果数据区中有多个数据,则在对其他数据操作时需修改地址。例如:

```
NUM DB 11H,22H,33H
...
MOV BL,NUM                     ;将 11H送 BL
MOV AL,NUM+2                   ;将 33H送 AL
```

4. 表达式

汇编语言语句中的表达式(Expression)不是指令,本身不能执行。在程序汇编时,汇编程序将表达式进行相应的运算,得出一个确定的值。所以在程序执行时,表达式本身已是一个有确定值的操作数。表达式仅是将求其值的计算任务交给了汇编程序来完成。

表达式中常用的运算符有以下几种。

1) 算术运算符

表达式中常用的算术运算符有+、-、*、/和 MOD(取余数)等。当算术运算符用于数值表达式时,其汇编结果是一个数值。例如:

```
MOV AL,8+5
```

等价于

```
MOV AL,13
```

当算术运算符用于地址表达式时,通常只使用其中的"+"和"-"两种运算符。例如,VAR+2 表示变量 VAR 的地址加上 2 得到新的存储单元地址。

【例 4-2】 将字数组 NUM 的第 8 个字送累加器 AX。指令为

```
MOV AX,NUM+(8-1)*2
```

这里,NUM 代表数组的首地址,(8-1)*2 是第 8 个字相对首地址的位移量。

2) 逻辑运算符

逻辑运算符包括 AND、OR、NOT 和 XOR。逻辑运算符只用于数值表达式,用来对数值进行按位逻辑运算并得到一个数值结果。例如:

```
MOV AL,0ADH AND 0CCH
```

等价于

```
MOV AL,8CH
```

请注意,不要把逻辑运算符 AND、OR、XOR、NOT 与同名称的 CPU 指令相混淆。

3) 关系运算符

关系运算符共有 6 个:EQ(等于)、NE(不等于)、LT(小于)、GT(大于)、LE(小于等于)、GE(大于等于)。参与关系运算的必须是两个数值或同一段中的两个存储单元地址,运算结果是一个逻辑值。当关系不成立(为假)时,结果为 0;关系成立(为真)时,结果为 0FFFFH。例如:

```
MOV AX,4 EQ 3             ;关系不成立,汇编成指令 MOV AX,0
```

```
MOV AX,4 NE 3             ;关系成立,汇编成指令 MOV AX,0FFFFH
```

4）取值运算符和属性运算符

取值运算符用来分析一个存储器操作数的属性，而属性运算符则可以规定存储器操作数的某个属性。

这里介绍常用的两个取值运算符 OFFSET 和 SEG 及属性运算符 PTR。

（1）OFFSET。利用运算符 OFFSET 可以得到一个标号或变量的偏移地址。例如：

```
MOV SI,OFFSET DATA1                ;将变量 DATA1 的偏移地址送 SI
```

这条指令与下边的指令执行结果相同。

```
LEA SI,DATA1                       ;取 DATA1 的偏移地址送 SI
```

（2）SEG。利用运算符 SEG 可以得到一个标号或变量的段地址。例如：

```
MOV AX,SEG DATA                    ;将变量 DATA 的段地址送 AX
MOV DS,AX                          ;DS←AX
```

（3）PTR。属性运算符用来指定位于其后的存储器操作数的类型。例如：

```
CALL DWORD PTR[BX]                 ;说明存储器操作数为 4 个字节长,即调用远过程
MOV AL,BYTE PTR[SI]                ;将 SI 指向的一个字节数送 AL
```

如果一个变量已经定义为字变量，利用 PTR 运算符可以修改它的属性。例如，变量 VAR 已定义为字，现要将 VAR 当作字节操作数使用，则

```
MOV AL,VAR                         ;指令非法,因为两操作数字长不相等
MOV AL,BYTE PTR VAR                ;指令合法,BYTE PTR 强制将 VAR 变为字节操作数
```

PTR 运算符仅对当前指令有效。

5）其他运算符

（1）方括号“[ ]”。指令中用方括号表示存储器操作数，方括号里的内容表示操作数的偏移地址。

（2）段重设运算符“:”。运算符“:”（冒号）跟在某个段寄存器名（DS、ES、SS）之后表示段重设，用来指定一个存储器操作数的段属性而不管其原来隐含的段是什么。例如：

```
MOV AX,ES:[DI]                     ;把 ES 段中由 DI 指向的字操作数送 AX
```

4.2 伪 指 令

指示性语句中的伪操作命令，无论表示形式或其在语句中所处的位置都与 CPU 指令相似，因此也称为伪指令，但两者之间有着重要的区别。

首先，CPU 指令在程序运行时由 CPU 执行，每条指令对应 CPU 的一种特定的操作，如数据传送、算术运算等；而伪操作命令在汇编过程中由汇编程序执行，如定义数据、分配

存储区、定义段以及定义过程等。其次,汇编以后,每条 CPU 指令都被汇编并产生一条与之对应的目标代码,而伪操作则不产生与之相应的目标代码。

宏汇编程序 MASM 提供了几十种伪操作,限于篇幅,这里只介绍几种常用的伪操作指令。

4.2.1 数据定义伪指令

数据定义伪指令用来定义变量的类型、给变量赋初值或给变量分配存储空间。

1. 格式

数据定义伪指令的一般格式为

[变量名] 伪操作 操作数[,操作数 …]

方括号中的变量名为可选项,变量名后面不跟冒号。常用的数据定义伪指令有以下 5 种。

(1) DB(Define Byte):定义变量为字节类型。变量中的每个操作数占一个字节(0～0FFH)。DB 伪指令也常用来定义字符串。

(2) DW(Define Word):定义变量为字类型。DW 伪指令后面的每个操作数都占用 2 个字节。在内存中存放时,低字节在低地址,高字节在高地址。

(3) DD(Define Double Word):用来定义双字类型的变量。DD 伪指令后面的每个操作数都占用 4 个字节。在内存中存放时,同样是低字节在低地址,高字节在高地址。

(4) DQ(Define Quad Word):定义四字(QWORD,8 个字节)类型的变量。在内存中存放时,低字节在低地址,高字节在高地址。

(5) DT(Define Ten Bytes):定义十字节(TBYTE)类型的变量。DT 伪操作后面的每个操作数都为 10 个字节的压缩 BCD 数。

2. 操作数

数据定义伪操作后面的操作数可以是常数、表达式或字符串。一个数据定义伪指令可以定义多个数据元素,但每个数据元素的值不能超过由伪操作所定义的数据类型限定的范围。例如,DB 伪指令定义数据的类型为字节,则所定义的数据元素的范围为 0～255(无符号数)或－128～＋127(有符号数)。字符和字符串都必须放在单引号中。超过两个字符的字符串只能用于 DB 伪指令。例如:

```
DATA  DB  11H,33H                  ;定义包含两个元素的字节变量 DATA
NUM   DW  100*5+88                 ;定义一个字类型变量 NUM,其初值为表达式的值
STR   DB  'Hello! '                ;定义一个字符串,字符串的首地址为 STR
SUM   DQ  0011223344556677H        ;将 4个字存入变量 SUM。它们在内存中的存放
                                   ;地址由低到高分别为:77H、66H、55H、44H、
                                   ;33H、22H、11H、00H
ABC   DT  1234567890H              ;将一个 10字节的压缩 BCD数赋给变量 ABC
```

```
                        ;它们在内存中的存放地址由低到高分别为:
                        ;90H、78H、56H、34H、12H、00H、00H、00H、
                        ;00H、00H
```

数据定义伪操作的操作数除以上几种外,还可以是问号“?”。“?”在这里的作用是给变量保留相应的存储单元,而不赋予变量确定的值。

例如:

```
DATA2 DW ?              ;为变量 DATA2 分配 2 个字节的空间,初值为任意值
```

3. 重复操作符

当同样的操作数重复多次时,可用重复操作符“DUP”表示。DUP 的一般格式为

```
[变量名] 数据定义伪操作 n DUP(初值[,初值 …])
```

圆括号中为重复的内容,n 为重复次数。如果用“n DUP(?)”作为数据定义伪操作的唯一操作数,则汇编程序仅保留 n 个元素大小的数据区。数据区中的初始值为任意值。例如:

```
DATA1 DB 20 DUP(?)          ;为变量 DATA1 分配 20 个字节的空间,初值为任意值
DATA3 DB 20 DUP(30H)        ;为变量 DATA3 分配 20 个字节空间,初值均为 30H
```

重复操作符主要应用于需要预留存储区域且对其初始值不关心的场合,如定义堆栈区、为数据定义缓冲区等。

【例 4-3】 画图表示下列变量在内存中的存放顺序。

```
VAR1 DB 11H,'HELLO! '
VAR2 DW 12H,3344H
VAR3 DD 1234H
VAR4 DW 2 DUP(88H)
VAR5 DB 2 DUP(56H,78H)
```

以上各变量在内存中的存放顺序如图 4-1 所示。

4.2.2 符号定义伪指令

在程序中,有时会多次出现同一个表达式,为了方便起见,常将该表达式赋予一个名字,以后凡是用到该表达式的地方就用这个名字来代替。在需要修改该表达式的值时,只需在赋予名字的地方修改即可。

符号定义伪指令 EQU 就是用于给某个表达式赋予一个名字或者说是使某个字符名等于某个表达式的值。符号定义伪指令的一般格式为

```
名字 EQU 表达式
```

格式中的表达式可以是一个常数、符号、数值表达式、地址表达式甚至可以是指令助

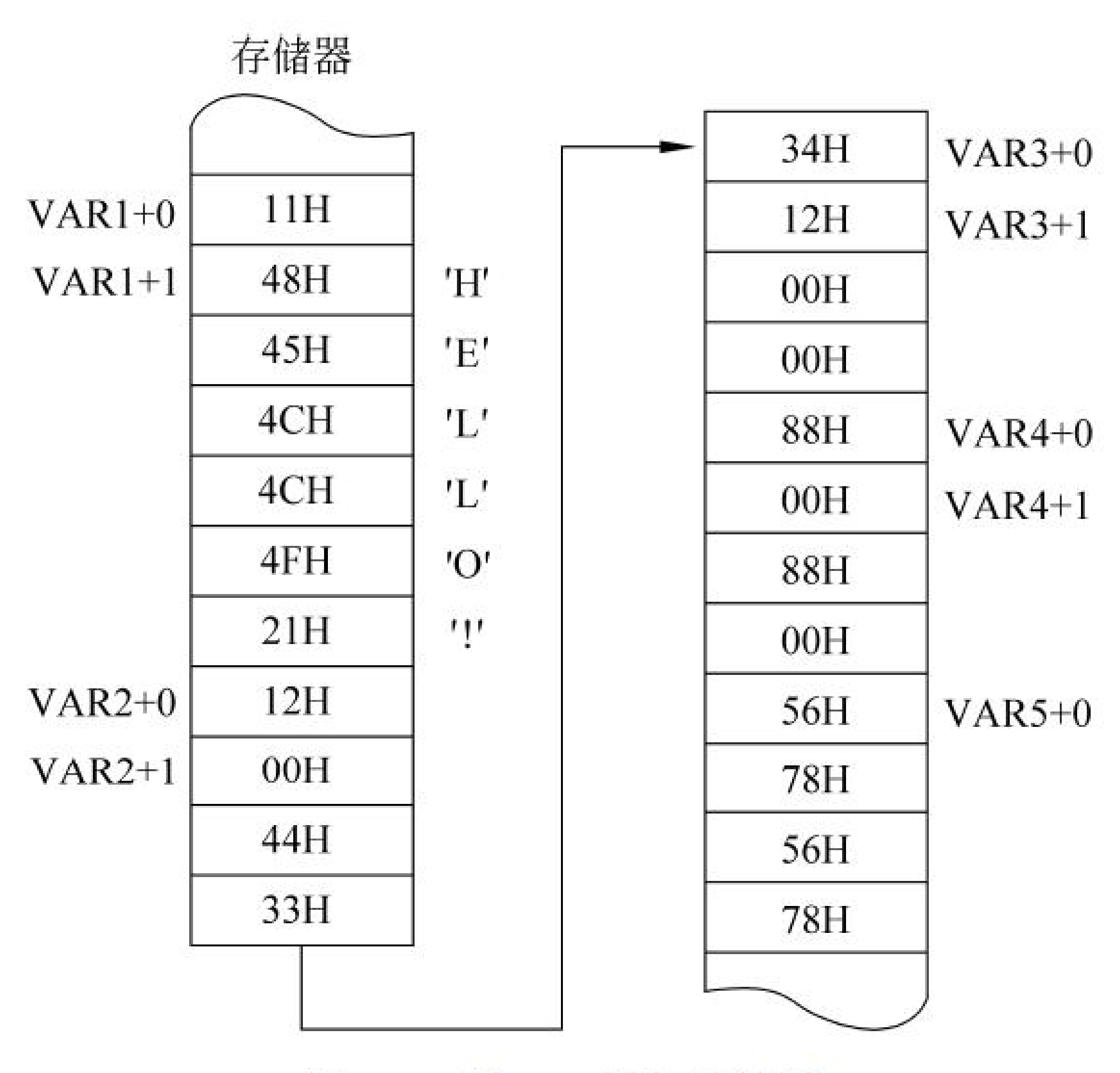


图 4-1　例 4-3 变量定义图

记符。例如：

```
CR    EQU   0DH              ;表示 CR=0DH
TEN   EQU   0AH              ;表示 TEN=0AH
VAR   EQU   TEN*2+1024       ;表示 VAR=伪操作后边表达式的值
ADR   EQU   ES:[BP+DI+5]     ;地址表达式
```

在程序段中应用以上的定义。

```
MOV   AL,TEN                 ;AL←0AH
CMP   AL,TEN                 ;AL的内容与 0AH进行比较
GOTO  WORD PTR ADR           ;转到以字单元 ES:[BP+DI+5]的内容为地址的程序段执行
```

利用 EQU 伪指令可以用一个名字代表一个数值或用一个较简短的名字代表一个较长的名字等，但不允许用 EQU 对同一个符号重复定义。若希望对一个符号重复定义，可用“＝”伪指令。例如：

```
FACTOR = 10H             ;FACTOR代表了数值 10H
…
FACTOR = 25H             ;从现在开始，FACTOR代表了数值 25H
```

4.2.3　段定义伪指令

前边已经讲过，汇编语言源程序是用分段的方法来组织程序、数据和变量的。一个源程序由若干个逻辑段组成。段定义伪指令用来定义汇编语言源程序中的逻辑段。其格式为

```
段名 SEGMENT [定位类型] [组合类型] ['类别']
```

⋮
段名 ENDS

源程序中的每个逻辑段由 SEGMENT 语句开始，到 ENDS 语句结束。二者总是成对出现，缺一不可。中间省略的部分称为段体。对数据段、附加段和堆栈段来说，段体一般为变量、符号定义等伪指令；对代码段则是程序代码。SEGMENT 和 ENDS 前面的段名表示定义的逻辑段的名字，必须相同，否则汇编程序将无法辨认。起什么名字可由程序员自行决定，但不要与指令助记符或伪指令等保留字重名。后面方括号中为可选项，规定了该逻辑段的一些其他特性，下面分别加以介绍。

1. 定位类型

定位类型(Align)告诉汇编程序如何确定逻辑段的地址边界。定位类型有 4 种，如下所示。

(1) PARA(Paragraph)：说明逻辑段从一个节的边界开始。16 个字节称为一个节，所以段的起始地址应能被 16 整除，也就是段起始物理地址应为××××0H。在省略情况下，定位类型默认为 PARA。

(2) BYTE：说明逻辑段从字节边界开始，即可以从任何地址开始。此时本段的起始地址紧接在前一个段的后面。

(3) WORD：说明逻辑段从字边界开始，即本段的起始地址必须是偶数。

(4) PAGE：说明逻辑段从页边界开始。256 字节称为一页，故本段的起始物理地址应为×××00H。

2. 组合类型

组合类型(Combine)主要用在具有多个模块的程序中。组合类型用于告诉汇编程序，当一个逻辑段装入存储器时它与其他段如何进行组合。组合类型共有以下 6 种。

(1) NONE：表示本段与其他逻辑段不组合，即对不同程序模块中的逻辑段，即使具有相同的段名，也分别作为不同的逻辑段装入内存而不进行组合。默认情况下，组合类型是 NONE。

(2) PUBLIC：表示对于不同程序模块中用 PUBLIC 说明的具有相同段名的逻辑段，汇编时将它们组合在一起，构成一个大的逻辑段。

(3) STACK：组合类型为 STACK 时，其含义与 PUBLIC 基本一样，但仅限于作为堆栈的逻辑段使用，即在汇编时，将不同程序模块中用 STACK 说明的同名堆栈段集中成为一个大的堆栈段，由各模块共享。堆栈指针 SP 指向这个大的堆栈区的栈顶(最高地址+1)处。

(4) COMMON：表示对于不同程序模块中用 COMMON 说明的同名逻辑段，连接时从同一个地址开始装入，即各个逻辑段重叠在一起。连接之后的段长度等于原来最长的逻辑段的长度。重叠部分的内容是最后一个逻辑段的内容。

(5) MEMORY：表示当几个逻辑段连接时，本逻辑段定位在地址最高的地方。如果被连接的逻辑段中有多个段的组合类型都是 MEMORY，则汇编程序只将首先遇到的段

作为 MEMORY 段，而其余的段均当作 COMMON 段处理。

(6) AT 表达式：表示本逻辑段根据表达式求值的结果定位段地址。例如 AT 8000H 表示本段的段地址为 8000H，即本段的起始物理地址为 80000H。

3. 类别

类别(Class)是用单引号括起来的字符串，如代码段('CODE')、堆栈段('STACK')等，当然也可以是其他名字。设置类别的作用是，当几个程序模块进行连接时，将具有相同类别名的逻辑段装入连续的内存区内，类别名相同的逻辑段按出现的先后顺序排列；没有类别名的逻辑段与其他无类别名的逻辑段一起连续装入内存。

上述 3 个可选项主要用于多个程序模块的连接。若程序只有一个模块，即只包括代码段、数据段、附加段和堆栈段时，除堆栈段建议用组合类型 STACK 说明外，其他段的组合类型及类别均可省略。定位类型一般采用默认值 PARA。

【例 4-4】 将两个模块中的同名段进行组合。

模块 1：

```
STACK   SEGMENT STACK
      DB 100 DUP(0)
STACK   ENDS

DATA    SEGMENT COMMON
AREA1   DB 1024 DUP(0)
DATA    ENDS

CODE    SEGMENT PUBLIC
⋮
CODE    ENDS
```

模块 2：

```
STACK   SEGMENT STACK
      DB 50 DUP(0)
STACK   ENDS
DATA    SEGMENT COMMON
AREA1   DB 8192 DUP(0)
DATA    ENDS
CODE    SEGMENT PUBLIC
           ⋮
CODE    ENDS
END
```

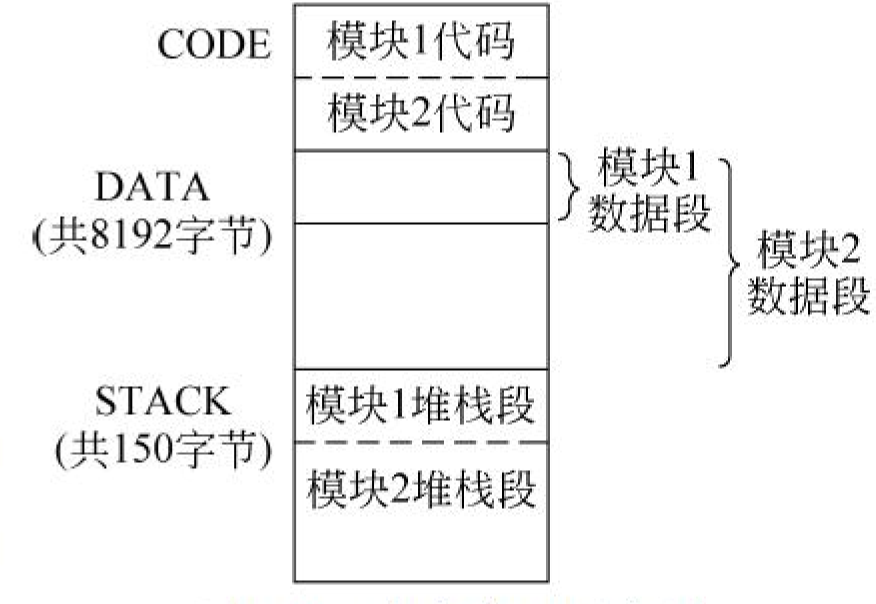


图 4-2 内存分配示意图

汇编连接后，存储器中的分配情况如图 4-2 所示。这里，两个模块中的代码段的名字相同，组合类型为 PUBLIC，故将它们连接成一个大的代

码段;数据段的名字也相同,用 COMMON 说明,则将它们重叠。因为模块 2 的数据段比模块 1 的长,所以数段长度为 8192 字节;同理,堆栈段组合成为一个大的堆栈区,共 150 个字节。

4.2.4 设定段寄存器伪指令

ASSUME 伪指令用于向汇编程序说明所定义的逻辑段属于何种类型的逻辑段。说明的方法是将逻辑段的段名与对应的段寄存器联系起来。该伪操作的一般格式为

```
ASSUME 段寄存器名:段名[,段寄存器名:段名[,…]]
```

格式中的段寄存器名可以是 CS、DS、ES 或 SS。

8088 的存储器采用分段结构,每个逻辑段最大可以是 64KB,且可有多个逻辑段。但每个程序模块最多只允许有 4 个逻辑段,即一个代码段、一个数据段、一个附加段和一个堆栈段。ASSUME 伪指令用来告诉汇编程序当前正在使用的各段的名字,换句话说,就是告诉汇编程序用 SEGMENT 伪操作定义过的段的段地址将要存放在哪个段寄存器中。但系统除了能够自动将代码段的段地址放入段寄存器 CS 之外,其他逻辑段的段地址都需要由程序员自己装入相应的段寄存器中,这个过程称为段寄存器的初始化。这样,当汇编程序汇编一个逻辑段时,即可利用相应的段寄存器寻址该逻辑段中的指令或数据。

在源程序中,ASSUME 伪指令放在可执行程序开始位置的前面,来看下面的例子。

【例 4-5】 ASSUME 伪指令的应用。

```
…
CODE  SEGMENT PARA PUBLIC 'CODE'
      ASSUME CS:CODE,DS:DATA,ES:EDATA,SS:STACK
      MOV AX,DATA
      MOV DS,AX                   ;将数据段的段地址送入 DS
      MOV AX,EDATA
      MOV ES,AX                   ;将附加段的段地址送入 ES
      MOV AX,STACK
      MOV SS,AX                   ;将堆栈段的段地址送入 SS
      ⋮
CODE  ENDS
```

这就是一个完整的代码段的定义方法。汇编时,系统自动将代码段的段地址装入段寄存器 CS,所以代码段不需要在程序中初始化。但若定义了数据段、附加段和堆栈段,就需要程序员用指令把 DS、ES、SS 初始化(本程序段中假设已定义了数据段 DATA、附加段 EDATA 及堆栈段 STACK)。

4.2.5 过程定义伪指令

程序设计中,通常将具有某种功能的程序段看作一个过程(即子程序),它可以被别的

程序调用(CALL)。过程定义伪指令的一般格式为

```
过程名    PROC [NEAR/FAR]
             ⋮
             RET
过程名 ENDP
```

过程名实际上是过程入口的符号地址,PROC 和 ENDP 前的过程名必须相同。它们之间的部分是过程体,过程体内至少要有一条返回指令 RET,以便在程序调用结束后能返回原地址。过程可以是近过程(与调用程序在同一个代码段内),此时伪操作 PROC 后的类型是 NEAR,但可以省略;若过程为远过程(与调用程序在不同的代码段内),则伪操作 PROC 后的类型是 FAR,不能省略。过程可以嵌套,即一个过程可调用另一个过程;过程也可以递归,即过程可以调用过程本身。例如:

```
NAME1 PROC FAR
           ⋮
  CALL NAME2
           ⋮
        RET
  NAME2 PROC  ┐
           ⋮  │
        RET   ├ 过程 NAME2 嵌入在过程 NAME1 中
  NAME2 ENDP  ┘
NAME1 ENDP
```

【例 4-6】 编写一个 10ms 延时的子程序。

```
DELAY   PROC                ;定义一个近过程
        PUSH BX             ;保护 BX 原来的内容
        PUSH CX             ;保护 CX 原来的内容
        MOV BL,2            ;外循环次数
NEXT:   MOV CX,4167         ;内循环次数 (实现延时 5ms)
W10MS:  LOOP W10MS          ;CX≠0 则循环
        DEC BL              ;修改外循环计数值
        JNZ NEXT            ;BX≠0 则进行第 2 轮循环
        POP CX              ;恢复 CX 原来的内容
        POP BX              ;恢复 BX 原来的内容
        RET                 ;过程返回
DELAY   ENDP                ;过程结束
```

4.2.6 宏命令伪指令

在汇编语言源程序中,如果需要多次使用同一个程序段,可以将这个程序段定义为一

个宏指令，然后每次需要时即可简单地用宏指令名来代替（称为宏调用），以避免重复书写，使源程序更加简洁、易读。

宏命令伪指令的格式为

```
宏命令名  MACRO [形式参数,…]
          (宏定义体)
          ENDM
```

宏命令名与过程名类似，是宏定义的标志，它位于宏操作符 MACRO 之前，但宏定义结束符前不加宏命令名。对宏命令名的规定与对标号的规定一样。

宏定义中的形式参数是任选的，可以只有一个，可以有多个，也可以没有。有多个参数时，各参数间要用逗号隔开。中间省略部分是实现某些操作的宏定义体。

在宏调用时，用实际参数顺序代替形式参数，若实际参数比形式参数多，则多余的实际参数被忽略。

【例 4-7】 两个数之和的宏定义和宏调用。

宏定义为

```
DADD  MACRO X,Y,Z
      MOV AX,X
      ADD AX,Y
      MOV Z,AX
      ENDM
```

这里，X、Y、Z 是形式参数。在源程序中调用宏 DADD 时可写为

```
DADD DATA1,DATA2,SUM
```

这里，DATA1、DATA2、SUM 是实际参数，在调用时 X、Y、Z 将被这 3 个实际参数替换。事实上，该宏命令汇编后对应的源程序为（这称为宏展开）

```
MOV AX,DATA1
ADD AX,DATA2
MOV SUM,AX
```

显然，宏调用与过程（子程序）调用有类似的地方。但这两种编程方法在使用上是有差别的，具体如下。

(1) 宏命令伪指令由宏汇编程序 MASM 在汇编过程中进行处理，在每个宏调用处，MASM 都用其对应的宏定义体替换。而调用指令 CALL 和返回指令 RET 则是 CPU 指令，执行 CALL 指令时，CPU 使程序的控制转移到子程序的入口地址。

(2) 宏指令简化了源程序，但不能简化目标程序。汇编以后，在宏定义处不产生机器代码，但在每个宏调用处，通过宏扩展，宏定义体的机器代码仍然出现多次，因此并不节省内存单元。而对于子程序，在目标程序中，定义子程序的地方将产生相应的机器代码，每次调用时只需用 CALL 指令，不再重复出现子程序的机器代码，因此可使目标程序较短，节省了内存单元。

(3) 从执行时间来看,调用子程序和从子程序返回需要保护断点、恢复断点等,都将额外占用 CPU 的时间,而宏指令则不需要,因此相对来说宏指令执行速度较快。可以这样说,宏指令是用空间换取了时间,而子程序是用时间换取了空间。

但无论如何,宏指令和子程序都是简化编程的有效手段。

4.2.7 模块定义与连接伪指令

在编写较大的汇编语言程序时,通常将其划分为几个独立的源程序(或称模块),然后将各个模块分别进行汇编,生成各自的目标程序,最后将它们连接成为一个完整的可执行程序。

在每一个模块的开始,常用伪指令 NAME 或 TITLE 为该模块定义一个名字,而在模块的结尾处要加结束伪指令 END,以使汇编程序结束汇编。

下面分别来看一下这 3 条伪指令的格式及操作。

1. NAME 伪指令

指令格式:

```
NAME 模块名
```

NAME 伪指令用于给汇编后得到的目标程序一个名字。NAME 伪指令的前面不允许再加标号,例如下面的语句是非法的。

```
BEGIN:NAME 模块名
```

2. TITLE 伪指令

TITLE 伪指令为程序清单的每一页指定打印的标题。其格式为

```
TITLE 标题名
```

标题名最多允许 60 个字符。如果程序中没有 NAME 伪指令,则汇编程序将 TITLE 伪指令后面的“标题名”中的前 6 个字符作为模块名;如果源程序中既没有使用 NAME,也没有使用 TITLE 伪操作,则汇编程序将源程序的文件名作为目标程序的模块名。

3. END 伪指令

END 伪指令表示源程序到此结束,指示汇编程序停止汇编。其格式为

```
END [标号]
```

END 伪操作后面的标号表示程序执行的开始地址。END 伪指令将标号的段值和偏移地址分别提供给 CS 和 IP 寄存器。标号是任选项,也可以没有。如果在 END 伪指令后没指定标号,则汇编程序把程序中第一条指令的地址作为程序执行的开始地址。如果有多个模块连接在一起,则只有主模块的 END 语句允许使用标号。

以上介绍了汇编语言中常用的各类伪指令及它们在源程序中的应用，下面来看一个定义了数据段和代码段的、具有完整程序结构的汇编语言源程序例。

【例 4-8】 求从 TABLE 开始的 10 个无符号字节数的和，结果放 SUM 字单元中。

```
DATA    SEGMENT                               ;定义数据段
TABLE   DB 12H,23H,34H,45H,56H                ;10个加数
        DB 67H,78H,89H,9AH,0FDH
SUM     DW ?
DATA    ENDS
;
CODE    SEGMENT                               ;定义代码段
        ASSUME CS:CODE,DS:DATA,ES:DATA
START:  MOV AX,DATA
        MOV DS,AX                             ;初始化 DS
        MOV ES,AX                             ;初始化 ES
        LEA SI,TABLE                          ;SI 指向 TABLE
        MOV CX,10                             ;循环计数器
        XOR AX,AX                             ;AX为中间结果
NEXT:   ADD AL,[SI]                           ;把一个数加到 AL 中
        ADC AH,0                              ;若有进位,则加到 AH 中
        INC SI                                ;指向下一个数
        LOOP NEXT                             ;若未加完,继续循环
        MOV SUM,AX                            ;若结束,存结果于 SUM
        HLT                                   ;结束
CODE    ENDS                                  ;代码段结束
        END START                             ;汇编结束,起始运行地址为 START
```

4.3 BIOS 和 DOS 功能调用

微型机的系统软件（如操作系统）提供了很多可供用户调用的功能子程序，包括控制台输入输出、基本硬件操作、文件管理、进程管理等。它们为用户的汇编语言程序设计提供了许多方便。用户可在自己的程序中直接调用这些功能，而无须再自行编写。

系统软件中提供的功能调用有两种：BIOS（Basic Input and Output System）功能调用（也叫低级调用）、DOS（Disk Opration System）功能调用（也称高级调用）。

BIOS 是被固化在计算机主机板上 Flash ROM 型芯片中的一组程序，与系统硬件有直接的依赖关系。在 IBM PC 的存储器系统中，BIOS 存放在地址为 0FE000H 开始的 8KB ROM（只读存储器）存储区域中，其功能包括系统测试程序、初始化引导程序、一部分中断矢量装入程序及外部设备的服务程序。使用 BIOS 提供的这些功能模块可以简化程序设计，使程序员不必了解硬件操作的具体细节，只要通过指令设置参数、调用 BIOS 功能程序，就可以实现相应的操作。

DOS 是 IBM PC 系列微机的操作系统(现在的 Pentium 系列微机仍能运行 DOS,而且最新的 Windows 操作系统也继续提供所有的 DOS 功能调用),负责管理系统的所有资源、协调微机的操作,其中包括大量的可供用户调用的服务程序。DOS 的功能调用不依赖于具体的硬件系统。

不论是 BIOS 功能调用还是 DOS 功能调用,用户程序在调用这些系统服务程序时,都不是使用 CALL 命令,而是采用软中断指令 INT n 来实现(故也称 BIOS 中断或 DOS 中断),这里的 n 表示中断类型码,不同的中断类型码表示不同的功能模块。由于不论是 DOS 功能还是 BIOS 功能,其每个功能模块中都包含了若干子功能,这些子功能用功能号来区分,在中断调用前需要将功能号装入 AH 寄存器。常用 DOS 和 BIOS 软中断功能见附录 D。

一般来讲,调用 DOS 或 BIOS 功能时,有以下几个基本步骤:①AH←功能号;②在指定寄存器中放入该功能所要求的入口参数;③执行 INT n 指令;④分析出口参数。

由于这些系统服务程序在系统启动时已被加载到内存中,程序入口也被放到了中断向量表中,因此用户程序不必与这些服务程序的代码连接。

使用 BIOS 或 DOS 功能调用会使编写的程序简单、清晰、可读性好而且代码紧凑、调试方便。

因篇幅所限,下面仅介绍几个最常用的 BIOS 和 DOS 中断。

4.3.1 BIOS 功能调用

BIOS 软中断简表见附录 C.4,包括屏幕显示、磁盘输入输出、键盘输入和打印机输出、异步通信控制、时钟控制等。以下简要介绍键盘输入和显示器输出功能。

1. 键盘输入

键盘是计算机最基本的输入设备,通常包括 3 种基本类型:字符键(如字母 A~Z、数字等)、扩展功能键(如 Home、End、Back Space、Del 等)以及和其他键组合使用的控制键(如 Alt、Ctrl、Shift 等)。

字符键给计算机传送一个 ASCII 码表示的字符,扩展功能键产生一个动作,如按下 End 键可使光标置于屏幕上文本的末尾,控制键能改变其他键所产生的字符码。

键盘上的每个键都对应了一个扫描码,扫描码用一个字节表示,低 7 位是数字编码,最高位(D_7)表示键的状态。当有键按下时,$D_7=0$;键放开时,$D_7=1$。根据扫描码就能唯一地确定哪个键改变了状态。

BIOS 键盘处理程序将获取的扫描码转换为相应的字符码。对大多数键,字符码就是 ASCII;对部分控制键(如 Alt、F1~F12),字符码为 0。转换后的字符码和扫描码存储在键盘缓冲区中。

BIOS 的键盘中断的类型码为 16H,送入 AH 的功能号可以是 0、1 或 2。

(1) 若只想取得按键的字符码和扫描码,可通过以下指令实现。

```
MOV AH,0
```

```
INT 16H
```

执行结果：AL＝字符码，AH＝扫描码。

(2) 若想判断有无键按下，可使用1号功能。

```
MOV AH,1
INT 16H
```

执行结果：若ZF＝0，则AL＝字符码，AH＝扫描码；若ZF＝1，则键盘缓冲区空。

2号功能用来判断Shift、Alt、Num等功能键是否被按下。其进一步的描述参见相关书籍。

【例4-9】 判断是否有控制键F8按下，若有则转NEXT。

题目分析：

获取按键的字符码和扫描码可调用类型码为16H的0号功能。通过查表可知F8的扫描码为42H。

程序如下：

```
        MOV AH,0
        INT 16H                         ;读取按键的字符码和扫描码
        CMP AL,0
        JNZ ERROR                       ;若不是控制键转 ERROR
        CMP AH,42H                      ;判断是否为 F8
        JE NEXT                         ;若是 F8则转 NEXT
          ⋮
NEXT:
          ⋮
ERROR:
          ⋮
```

2. 显示器输出

显示器通过显示适配器(Display Adaptor)与PC相连，显示适配器也称显卡，是计算机与显示器的接口，分为单色显示适配器(MDA，Monochrome Display Adaptor)和彩色图形适配器(CGA，Color Graphics Adaptor)。目前较为流行的图形适配器有EGA(Enhanced Graphics Adaptor)和VGA(Video Graphics Adaptor)以及在VGA基础上发展起来的SVGA(Super Video Graphics Adaptor)。

显示器的屏幕是由行和列组成的二维系统。每个字符都对应一个特定的行和列，0行0列表示屏幕的左上角。

BIOS显示器输出的类型码为10H，功能较强，主要包括设置显示方式、设置光标大小和位置、设置调色板号、显示字符和图形等。

对所有的显示适配器，文本方式下显示字符的原理都一样。对应屏幕上的每个字符，主存中都有相应的地址。每个字符在主存中占用两个字节单元，一个是字符的ASCII码，另一个是字符的属性(这里的属性是指显示的字符是否闪烁、何种颜色、是否亮度加

强等)。

若要显示一个字符,通常需要先设置光标位置(功能号为2),然后提供被显示字符的ASCII码及其属性,它们的功能号分别为9和10。这两个功能的共同特点是:在光标处显示字符且显示后光标不动。功能9既显示字符也显示其属性,功能10只显示字符,其属性值就是该位置上原有的属性。调用格式如下:

```
MOV AH,<功能号>
MOV BH,<页号>                ;对单色显示,显示页永远是0
MOV AL,<待显示字符>
MOV BL,<属性值>              ;对10号功能不需要
MOV CX,<重复显示次数>
INT 10H
```

【例4-10】 将光标置于0显示页的(20,30)位置,并以正常属性显示一个“$”。

题目分析:

置光标位置和字符显示均调用类型码为10H的BIOS中断。置光标位置的功能号是2,该功能要求将行、列参数分别送到DH和DL寄存器中。字符显示可使用功能号9。

程序如下:

```
MOV AH,2              ;置光标位置
MOV BH,0              ;对单色显示,显示页=0
MOV DH,20             ;行号
MOV DL,30             ;列号
INT 10H
MOV AH,9              ;显示字符及其属性
MOV BH,0              ;页号=0
MOV BL,7              ;属性设置(正常显示、黑色背景、白色字符)
MOV AL,'$'            ;送待显示字符
MOV CX,1              ;置重复次数
INT 10H
```

4.3.2 DOS功能调用

所有的DOS系统功能调用都是利用软中断指令INT 21H来实现的。也就是说,在程序中需要调用DOS功能的时候,只要使用一条INT 21H指令即可。INT 21H是一个具有90多个子功能的中断服务程序,这些子功能大致可以分为4个方面:设备管理、目录管理、文件管理和其他。其功能一览表见附录D.3。为了便于用户使用这些子功能,INT 21H对每一个子功能都进行了编号,称为功能号。这样,用户就能通过指定功能号来调用INT 21H的不同子功能。

DOS系统功能调用的使用方法如下:①AH←功能号;②在指定寄存器中放入该功能所要求的入口参数;③执行INT 21H指令;④分析出口参数。

下面介绍 INT 21H 的几个最常用的功能。

1. 键盘输入

键盘上的按键分为 3 种类型：①字符键，如字母、数字等；②功能键，如 Del、Enter 等；③组合键，如 Shift、Alt 等。

DOS 系统功能通过调用字符输入子功能，可以接收从键盘上输入的字符，输入的字符将以对应的 ASCII 码的形式存放。例如，若在键盘上按下数字键“9”，则键盘输入功能将返回一个字符 9 的 ASCII 码 39H。如果程序要求的是其他类型的值，则应自行编程进行转换。INT 21H 提供了若干支持键盘输入的子功能，这里只介绍单字符输入和字符串输入两种。

1）单字符输入

功能号 1、7 和 8 都可以接收键盘输入的单字符，输入的字符以 ASCII 码形式存放在累加器 AL 中。其中 7 号和 8 号功能无回显，1 号功能有回显（回显是指键盘输入的内容同时也显示在显示器上）。编程时，可根据输入的信息是否需要自动显示来选择三者之一。这些功能常用来回答程序中的提示信息，或选择菜单中的可选项以执行不同的程序段。

【例 4-11】 从键盘输入一个“Y”或“N”字符。

```
        ⋮
KEY: MOV AH,1                    ;有回显的键盘输入。功能号 1送(AH)
     INT 21H                     ;当按下键后,返回 AL=字符的 ASCII码
     CMP AL,'Y'                  ;比较输入的是否是 Y
     JE YES                      ;输入字符"Y"则转至 Yes语句处
     CMP AL,'N'                  ;比较输入的是否是 N
     JE NOT                      ;输入字符"N"则转至 NOT语句处
     JMP KEY                     ;输入其他字符,转至 KEY语句处,继续等待输入
YES:
        ⋮
NOT:
        ⋮
```

2）字符串输入

输入字符串可通过调用 DOS 功能的 0AH 号功能来实现。该功能要求用户指定一个输入缓冲区来存放输入的字符串。缓冲区一般定义在数据段，其定义格式有严格的要求，必须按照图 4-3 所示的结构。第一个字节为用户定义的缓冲区长度，若输入的字符数（包括回车符）大于此值，则喇叭会发出嘟嘟叫声，且光标不再右移直到输入回车符为止；

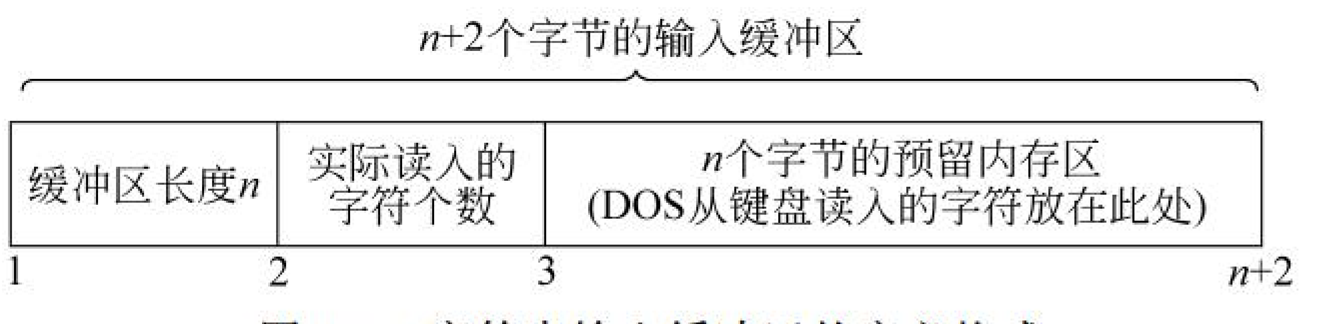


图 4-3 字符串输入缓冲区的定义格式

缓冲区第二个字节为实际输入的字符数(不包括回车符),由 0AH 号功能自动填入;DOS 从第三个字节开始存放输入的字符。显然,缓冲区的总长度等于缓冲区长度加 2。在调用本功能前,应把输入缓冲区的起始偏移地址预置入 DX 寄存器。

【例 4-12】 从键盘上输入字符串“HELLO”,并在串尾加结束标志'$'。

```
DATA SEGMENT
STRING DB 10,0,10 DUP(?)              ;定义缓冲区
DATA   ENDS
;
CODE   SEGMENT
ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       LEA DX,STRING                  ;缓冲区偏移地址送 DX
       MOV AH,0AH                     ;字符串输入功能号 0AH送 AH
       INT 21H                        ;从键盘输入字符串
       MOV CL,STRING+1                ;实际输入的字符个数送 CL
       XOR CH,CH
       ADD DX,CX                      ;得到字符串尾地址
       MOV BX,DX
       MOV BYTE PTR[BX+2],'$'         ;插入串结束符
       MOV AH,4CH                     ;返回 DOS
       INT 21H
       CODE ENDS
            END START
```

2. 显示器输出

在显示器(CRT)上显示的内容都是字符形式,如果是数字,则一定是其对应的 ASCII 码。例如,若要在显示器上显示 5,需要先将二进制的 5 转换为 5 的 ASCII 码 35H。

要将一个字符串送到显示器显示,可调用 DOS 功能的 2、6、9 号功能实现。其中,功能 2、6 用于显示单个字符,功能 9 显示一个字符串。

1) 单字符显示

用功能 2 显示一个字符的程序段如下:

```
…
MOV   DL,<要显示的字符>              ;要显示的字符必须放在 DL中
MOV   AH,2                          ;功能号送 AH
INT   21H                           ;执行系统功能调用
…
```

用功能 6 显示一个字符的程序段如下:

```
…
MOV   DL,<要显示的字符>              ;要显示的字符放在 DL中 (但不能是 0FFH)
```

```
MOV   AH,6                        ;功能号送 AH
INT   21H                         ;执行系统功能调用
…
```

【例 4-13】 在屏幕上依次显示"1""2""3""A""B""C"6 个字符。

```
DATA SEGMENT
STR DB '123ABC'
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX                  ;初始化段寄存器
       LEA BX,STR                 ;取字符变量的偏移地址
       MOV CX,6                   ;设循环次数
LPP:   MOV AH,2                   ;将功能号 2送 AH
       MOV DL,[BX]                ;取一个要显示的字符到 DL
       INC BX                     ;修改指针
       INT 21H                    ;调用中断 21H
       LOOP LPP
       MOV AH,4CH                 ;返回 DOS
       INT 21H
CODE   ENDS
       END START
```

2）字符串显示

要在显示器上显示字符串，可调用 DOS 功能的 9 号功能。9 号功能是 DOS 调用独有的，该功能要求被显示的字符串必须以'$'字符作为结束符，否则会引起屏幕混乱。显示时如果希望光标能自动换行，则应在字符串结束前加上回车及换行的 ASCII 码 0DH 和 0AH。

【例 4-14】 在屏幕上显示欢迎字符串"Hello，World！"。

```
DSEG SEGMENT
STRING DB 'Hello,World! ',0DH,0AH,'$ '         ;定义要显示的字符串
DSEG ENDS
CSEG SEGMENT
      ASSUME CS:CSEG,DS:DSEG
START: MOV AX,DSEG
       MOV DS,AX
       LEA DX,STRING                           ;获取要显示字符串的首地址
       MOV AH,09H                              ;调用字符串显示功能
       INT 21H
       MOV AH,4CH                              ;调用返回 DOS 功能
       INT 21H                                 ;返回 DOS
CSEG   ENDS
```

```
        END START
```

【例 4-15】 从键盘输入一串字符,在字符串尾插入'$',并显示该字符串。

```
DATA     SEGMENT
BUFSIZE  DB 50                       ;最多可输入 50个字符
ACTLEN   DB ?                        ;实际输入的字符数
CHARS    DB 50 DUP(20H)              ;实际输入的字符从此开始存放
DATA     ENDS
;
CODE     SEGMENT
         ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX
       MOV DX,OFFSET BUFSIZE         ;输入缓冲区起始偏移地址送 DX
       MOV AH,0AH
       INT 21H                       ;输入字符串并放入缓冲区
       XOR CX,CX
       MOV CL,ACTLEN                 ;取得输入的字符个数
       MOV DX,OFFSET CHARS           ;输入的字符串起始地址送 DX
       MOV BX,DX                     ;将字符串首地址送 BX
       ADD BX,CX                     ;得到字符串尾地址
       MOV BYTE PTR[BX],'$'          ;在字符串尾插入'$'
       MOV AH,09H                    ;字符串显示功能
       INT 21H                       ;显示输入的字符串
       MOV AH,4CH                    ;调用返回 DOS功能
       INT 21H                       ;返回 DOS
CODE ENDS
     END START
```

3. 返回到 DOS

一个实际可运行的用户程序在执行完后,应该返回到 DOS 提示符状态(简称为返回 DOS),简单地用 HLT 指令使 CPU 停止运行将无法把控制权交还给 DOS 操作系统。为了能使程序正常退出并返回 DOS,可使用 DOS 系统功能调用的 4CH 号功能。用 4CH 号功能返回 DOS 的程序段如下:

```
MOV  AH,4CH                ;功能号送 AH
INT  21H                   ;返回 DOS
```

4.4 汇编语言程序设计基础

在前面几节中已分别介绍了 8088/8086 CPU 的指令系统、汇编语言源程序的格式、伪操作指令以及 DOS 的功能调用等。汇编语言程序设计要求能够综合运用这些知识来

解决实际工程问题。本节将通过一些具体的实例说明汇编语言源程序的基本设计方法。

4.4.1 程序设计概述

1. 程序质量的评价标准

一个高质量的程序不仅应满足设计要求、实现预先设定的功能并能够正常运行，还应具备可理解性、可维护性和高效率等性能。衡量一个程序的质量通常有以下几个标准：①程序的正确性和完整性；②程序的易读性；③程序的执行时间和效率；④程序所占内存的大小。

编写一个程序首先要保证它的正确性，包括语法上和功能上；应尽量采用结构化、模块化的程序设计方法，每个模块由基本程序结构组成，完成一个基本的功能；为便于阅读、理解，并易于测试和维护，应在每个功能模块前添加一定的功能说明，在程序语句后添加相应的语句注释，对较大型的程序，还应有完整的文档资料和管理。另外，程序的响应时间、实时处理能力、输入输出方式和结果、内存占用大小及安全可靠性等，也都是非常重要的性能指标。

2. 程序设计的一般步骤

依照软件工程理论，汇编语言的程序设计与高级语言的程序设计一样可分为以下几个步骤。

(1) 通过对实际问题的分析抽象出系统数学模型，建立系统的模块结构图。

(2) 确定各程序模块的数据结构及算法。算法设计是非常重要的，对同一个问题可能有不同的算法，一个算法的好坏对程序执行的效率会有很大的影响(如对有序表的查表，线性查找和折半查找算法的区别很大)。

(3) 画程序流程图。流程图是算法的一种表示方法。

(4) 用指令或伪指令为数据和程序代码分配内存单元和寄存器，这是汇编语言程序设计的一个重要特点。

(5) 编写源程序并保存，形成源程序文件(.ASM)。

(6) 通过汇编生成目标代码文件(.OBJ)，同时完成静态的语法检查。

(7) 通过链接生成可执行文件(.EXE)。

(8) 程序调试，通过后可进行整个系统的测试。

3. 程序的基本结构

任何一个复杂的程序都是由简单的基本程序构成的，同高级语言类似，汇编语言程序的设计也常用到以下这样几种基本程序结构：顺序程序、分支程序、循环程序、子程序。

顺序程序是直线运动的，既无分支，也无循环或转移，是最简单的一种程序结构。

但总是沿直线运动的程序并不多，经常会碰到因不同的条件去执行不同的程序的情况，这就是所谓的分支程序。分支程序可以是双分支，也可以是多分支。

对于需要反复做同样工作的情况则用循环程序实现。循环结构可以缩短程序长度且便于维护，但循环程序中需要有循环准备、结束判断等指令，故执行速度要比顺序结构的程序略慢一些。

子程序又称过程，相当于高级语言中的函数或过程，是具有独立功能的模块。在程序设计中，为了便于编写、调试和修改，使程序结构尽量简单、清晰，增强可读性，常采用模块化的程序设计方法，即按功能将程序化分为一个个独立的模块，还可进一步根据具体的任务化分成小的子模块，每个模块都可单独编辑和编译，生成自己的源文件(.ASM 和 .OBJ)，然后通过链接形成一个完整的可执行文件。

4.4.2 节～4.4.6 节将通过举例进一步说明这几种基本程序结构的设计方法。

4.4.2 顺序程序

顺序程序是最常见、最基本的程序结构。CPU 按照指令的排列顺序逐条执行。

【例 4-16】 编写 S＝86H×34H－21H 的程序，式中的 3 个数均为无符号数。

题目分析：

(1) 有 3 个数参加运算，所以要定义 3 个源操作数，因它们的类型相同，题目中又没有要求分别存放，故只需定义一个字节类型变量来标识存放 3 个数的地址；

(2) 还需要定义一个变量来存放运算结果，因运算中有乘法，故结果应为 16 位，因而存放结果的变量应定义为字类型的变量；

(3) 运算中要用到乘法指令，因 3 个操作数为无符号数，所以乘法指令用 MUL。

该顺序程序的流程图如图 4-4 所示。其程序编写如下：

```
DATA   SEGMENT
NUM    DB 86H,34H,21H              ;定义源操作数
RESULT DW ?                        ;定义结果存放单元
DATA   ENDS
;
CODE   SEGMENT
       ASSUME CS:CODE,DS:DATA
START: MOV AX,DATA
       MOV DS,AX                   ;初始化数据段寄存器
       LEA SI,NUM                  ;NUM的偏移地址送 SI
       LEA DI,RESULT               ;RESULT偏移地址送 DI
       MOV AL,[SI]                 ;AL←86H
       MOV BL,[SI+1]               ;BL←34H
       MUL BL                      ;AX←86H*34H
       MOV BL,[SI+2]               ;BL←21H
       MOV BH,0                    ;BH←0
       SUB AX,BX                   ;AX←86H*34H-21H
       MOV [DI],AX                 ;结果 S送 RESULT单元
       MOV AH,4CH                  ;返回 DOS
```

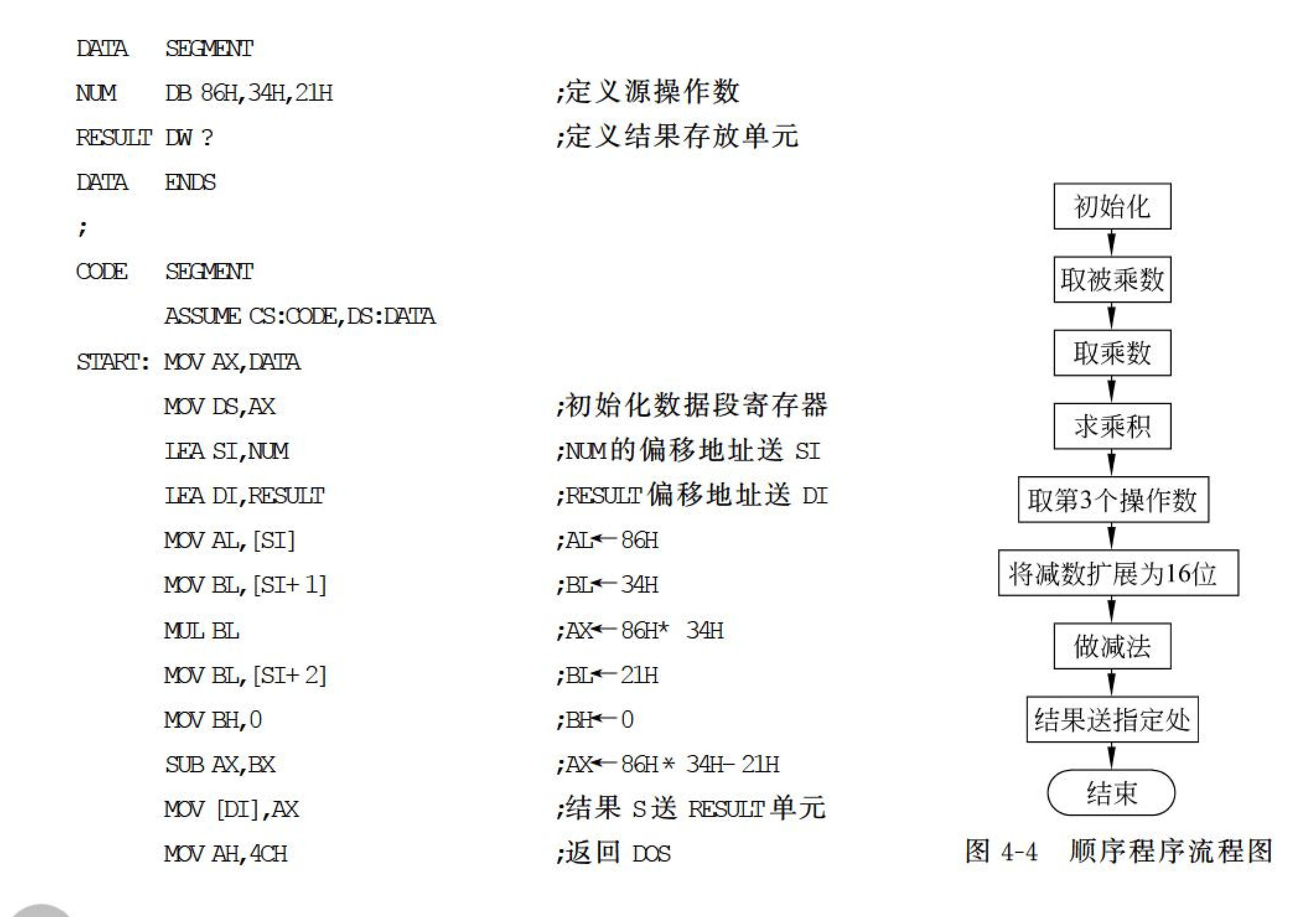


图 4-4 顺序程序流程图

```
        INT 21H
CODE    ENDS
        END START
```

【例 4-17】 内存自 TABLE 开始的连续 16 个单元中存放着 0～15 的平方值(称平方表),查表求 DATA 中任意数 $X(0\leqslant X\leqslant 15)$的平方值,并将结果放 RESULT 中。

题目分析:

由表的存放规律可知,表的起始地址与数 X 的和就是 X 的平方值所在单元的地址。程序编写如下:

```
DSEG    SEGMENT
TABLE   DB 0,1,4,9,16,25,36,49,64,81,
           100,121,144,169,196,225          ;定义平方表
DATA    DB ?
RESULT  DB ?                                ;定义结果存放单元
DSEG    ENDS
;
SSEG    SEGMENT STACK 'STACK'
DB 100 DUP(?)                               ;定义堆栈空间
SSEG ENDS
;
CSEG    SEGMENT
        ASSUME CS:CSEG,DS:DSEG,SS:SSEG
BEGIN:  MOV AX,DSEG                         ;初始化数据段
        MOV DS,AX
        MOV AX,SSEG                         ;初始化堆栈段
        MOV SS,AX
        LEA BX,TABLE                        ;置数据指针
        MOV AH,0
        MOV AL,DATA                         ;取待查数
        ADD BX,AX                           ;查表
        MOV AL,[BX]
        MOV RESULT,AL                       ;平方数存 RESULT 单元
        MOV AH,4CH
        INT 21H
CSEG    ENDS
        END BEGIN
```

4.4.3 分支程序

除最基本的顺序程序外,经常还会碰到根据不同的条件转移到不同的程序段执行的各种分支程序。分支程序的基本结构如图 4-5 所示,首先要判断条件是否成立,成立则执行程序段 P1,否则执行程序段 P2。这就是众所周知的 if-then 结构,如图 4-5(a)所示。程

序也可有多个分支，条件 1 成立则执行 P1；条件 2 成立则执行 P2；……；条件 n 成立则执行 Pn，如图 4-5(b)所示。这就是 if-then-else if 或 case 型程序结构。

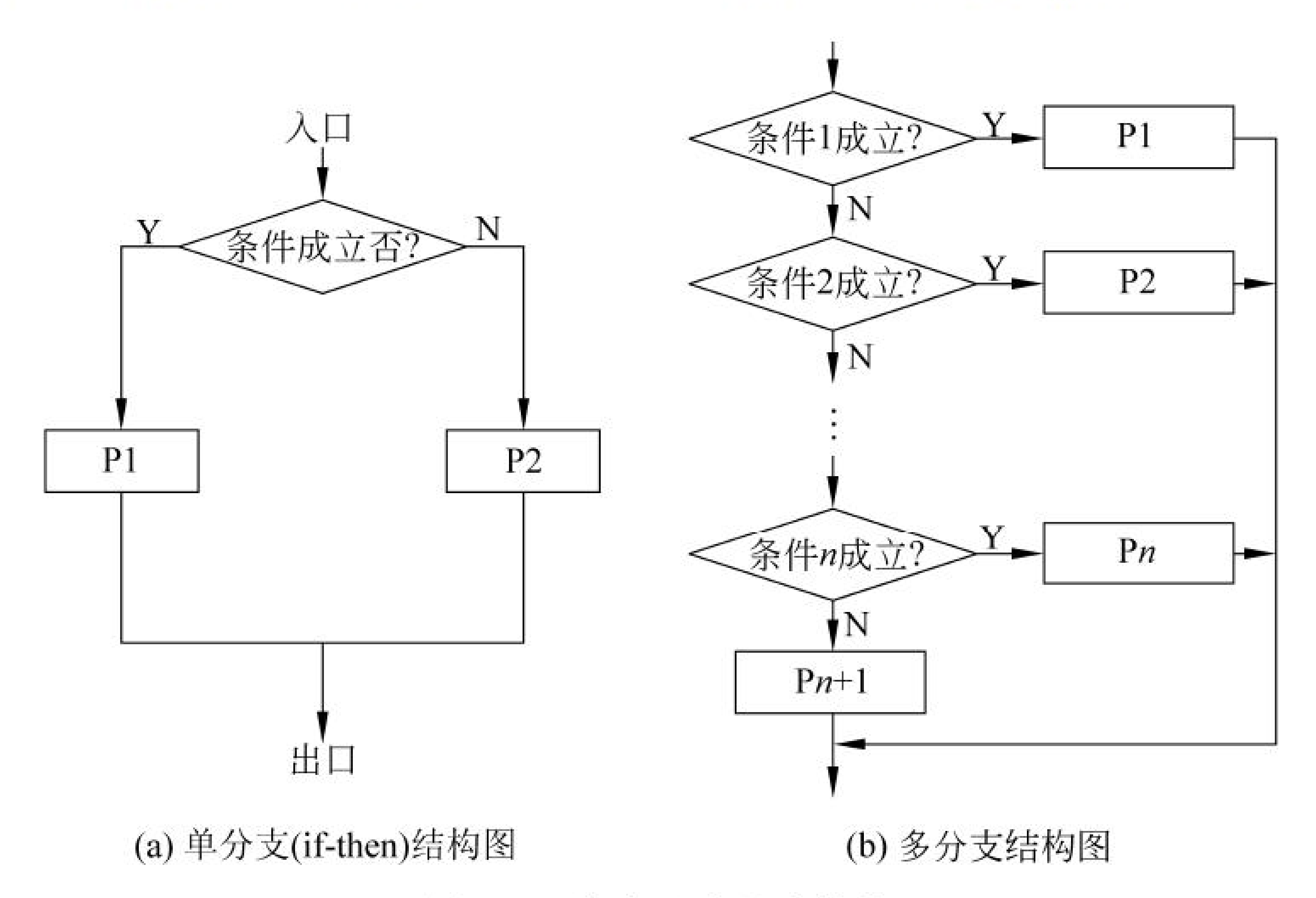


图 4-5　分支程序基本结构

【例 4-18】 编写程序，将数据区中以 BUFFER 为首地址的 100 个字节单元清零。

题目分析：

这是一个有两个分支的分支程序，结构如图 4-5(a)所示，将 00H 送到 BUFFER 起始的每个单元。每送一个字节判断一下计数值是否到 100，若不等于 100 则继续送，否则就结束，退出该程序段。

程序如下：

```
DATA   SEGMENT
BUFFER DB 100 DUP(?)
COUNT DW 100                        ;定义地址区长度
DATA   ENDS
;
STACK SEGMENT
       DW 32 DUP(?)
STACK ENDS
;
CODE   SEGMENT
       ASSUME CS:CODE,DS:DATA,SS:STACK
START: MOV AX,DATA
       MOV DS,AX                    ;初始化数据段
       MOV AX,STACK
       MOV SS,AX                    ;初始化堆栈段
       MOV CX,COUNT
       LEA BX,BUFFER
       ADD CX,BX
```

```
AGAIN: MOV BYTE PTR[BX],0                ;实现 100个单元清零
       INC BX
       CMP BX,CX
       JB AGAIN
       MOV AH,4CH
       INT 21H
CODE   ENDS
       END START
```

【例 4-19】 在当前数据段中 DATA1 开始的顺序 80 个单元中,存放着 80 位同学某门功课的考试成绩(0～100)。编写程序统计≥90 分、80～89 分、70～79 分、60～69 分以及<60 分的人数,并将结果放到同一数据段的 DATA2 开始的 5 个单元中。

题目分析:

(1) 这是一个具有多个分支的分支程序,结构如图 4-5(b)所示。需要将每一位学生的成绩依次与 90、80、70、60 进行比较,因是无符号数,所以用 CF 标志作为分支条件,相应指令为 JC;

(2) 由于对每一位学生的成绩都要进行判断,所以需要用循环来处理,每次循环处理一个学生的成绩(循环程序结构将在 4.4.4 节讲到);

(3) 因为无论成绩还是学生人数都不超过一个字节所能表示的数的范围,故所有定义的变量均为字节类型;

(4) 统计结果可用一个数组存放,元素 0 存放 90 分以上的人数,元素 1 存放 80 分以上的人数,元素 2 存放 70 分以上的人数,元素 3 存放 60 分以上的人数,元素 4 存放 60 分以下的人数。

程序如下:

```
DATA   SEGMENT
DATA1  DB 80 DUP(?)                      ;假定学生成绩已放入这 80个单元中
DATA2  DB 5 DUP(0)                       ;统计结果:≥90、80~89、70~79、60~69、<60
DATA   ENDS
;
CODE   SEGMENT
       ASSUME CS:CODE,DS:DATA
START: MOV  AX,DATA
       MOV  DS,AX
       MOV  CX,80                 ;统计人数送 CX
       LEA  SI,DATA1              ;SI 指向学生成绩
       LEA  DI,DATA2              ;DI 指向统计结果
AGAIN: MOV  AL,[SI]               ;取一个学生的成绩
       CMP  AL,90                 ;大于 90分吗?
       JC   NEXT1                 ;若不大于,继续判断
       INC  BYTE PTR[DI]          ;否则 90分以上的人数加 1
       JMP  STO                   ;转循环控制处理
NEXT1: CMP  AL,80                 ;大于 80分吗?
```

```
        JC    NEXT2              ;若不大于,继续判断
        INC   BYTE PTR[DI+1]     ;否则 80 分以上的人数加 1
        JMP   STO                ;转循环控制处理
NEXT2:  CMP   AL,70              ;大于 70 分吗?
        JC    NEXT3              ;若不大于,继续判断
        INC   BYTE PTR[DI+2]     ;否则 70 分以上的人数加 1
        JMP   STO                ;转循环控制处理
NEXT3:  CMP   AL,60              ;大于 60 分吗?
        JC    NEXT4              ;若不大于,继续判断
        INC   BYTE PTR[DI+3]     ;否则 60 分以上的人数加 1
        JMP   STO                ;转循环控制处理
NEXT4:  INC   BYTE PTR[DI+4]     ;60 分以下的人数加 1
STO:    INC   SI                 ;指向下一个学生成绩
        LOOP  AGAIN              ;循环,直到所有成绩都统计完
        MOV   AH,4CH             ;返回 DOS
        INT   21H
CODE    ENDS
        END   START
```

4.4.4 循环程序

当在程序设计中碰到某些需要多次重复执行的工作时,就可用循环程序来实现。如例 4-19 中,对每一个学生成绩的统计都要作同样的判断,故使用了循环结构。

循环程序在结构上包括循环初始化、循环体和循环控制 3 个部分。在形式上有两种:①先执行循环体,再判断条件看是否继续循环,如图 4-6(a)所示;②先检查条件是否满足,满足则执行循环体,否则就退出,如图 4-6(b)所示。

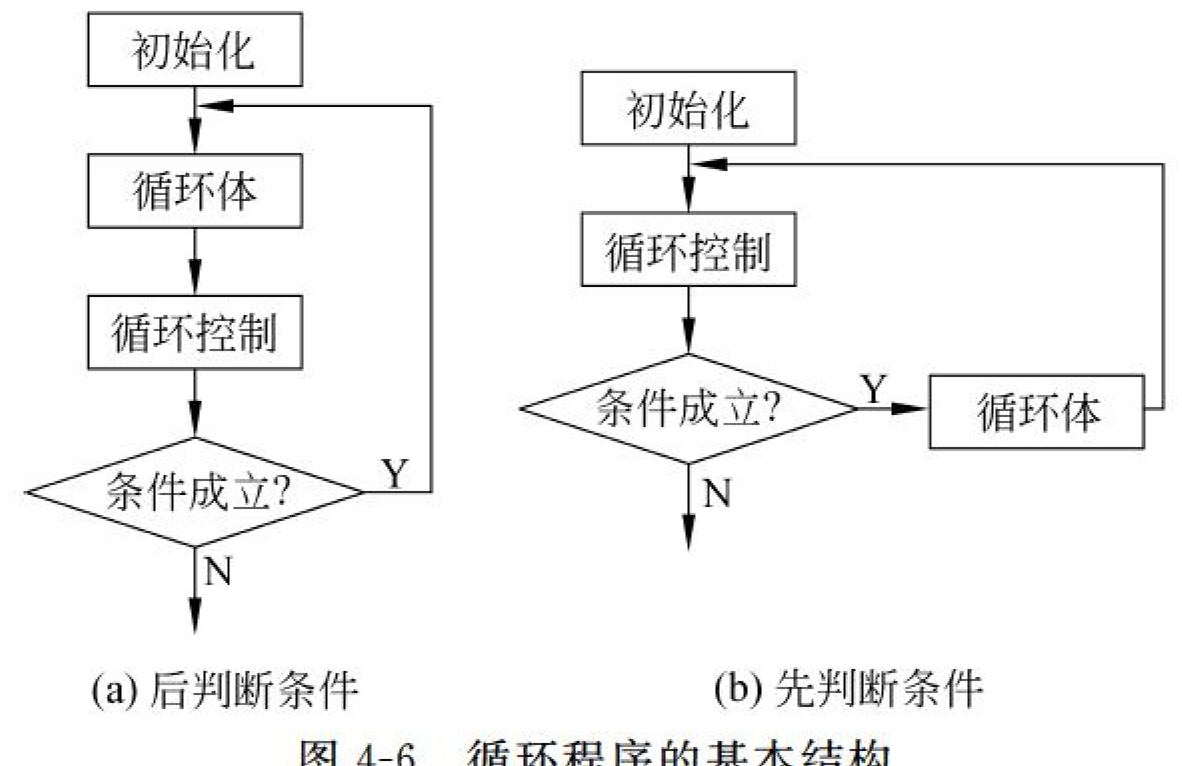


图 4-6　循环程序的基本结构

【例 4-20】 把从 MEM 单元开始的 100 个 16 位无符号数按从大到小的顺序排列。

题目分析:

(1) 这是一个排序问题,由于是无符号数的比较,可以直接用比较指令 CMP 和条件转移指令 JNC 来实现;

(2) 这是一个双重循环程序,先使第一个数与下一个数比较,若大于则使其位置保持不变,小于则将大数放低地址,小数放高地址(即两数交换位置);

(3) 以上完成了一次排序工作,再通过第二重的 99 次循环,即可实现对 100 个无符号数的大小排序。

程序编写如下:

```
DSEG    SEGMENT
MEM     DW 100 DUP(?)              ;假定要排序的数已存入这 100个字单元中
DSEG    ENDS
  ;
CSEG    SEGMENT
        ASSUME CS:CSEG,DS:DSEG
    START: MOV   AX,DSEG
           MOV   DS,AX
           LEA   DI,MEM      ;DI 指向待排序数的首址
           MOV   BL,99       ;外循环只需 99次即可

      ;外循环体从这里开始
    NEXT1: MOV   SI,DI       ;SI 指向当前要比较的数
           MOV   CL,BL       ;CL 为内循环计数器

           ;以下为内循环
    NEXT2: MOV   AX,[SI]     ;取第一个数 Ni
           ADD   SI,2        ;指向下一个数 Nj
           CMP   AX,[SI]     ;Ni≥Nj?
           JNC   NEXT3       ;若大于,则不交换
           MOV   DX,[SI]     ;否则,交换 Ni 和 Nj
           MOV   [SI-2],DX
           MOV   [SI],AX
    NEXT3: DEC   CL          ;内循环结束?
           JNZ   NEXT2       ;若未结束,则继续
           ;内循环到此结束

           DEC   BL          ;外循环结束?
           JNZ   NEXT1       ;若未结束,则继续
           ;外循环体结束

           MOV   AH,4CH      ;返回 DOS
           INT   21H
    CSEG   ENDS
           END   START
```

该循环程序属于图 4-6(a)所示的先执行循环体,再判断条件以决定是否循环的结构。其程序流程如图 4-7 所示。

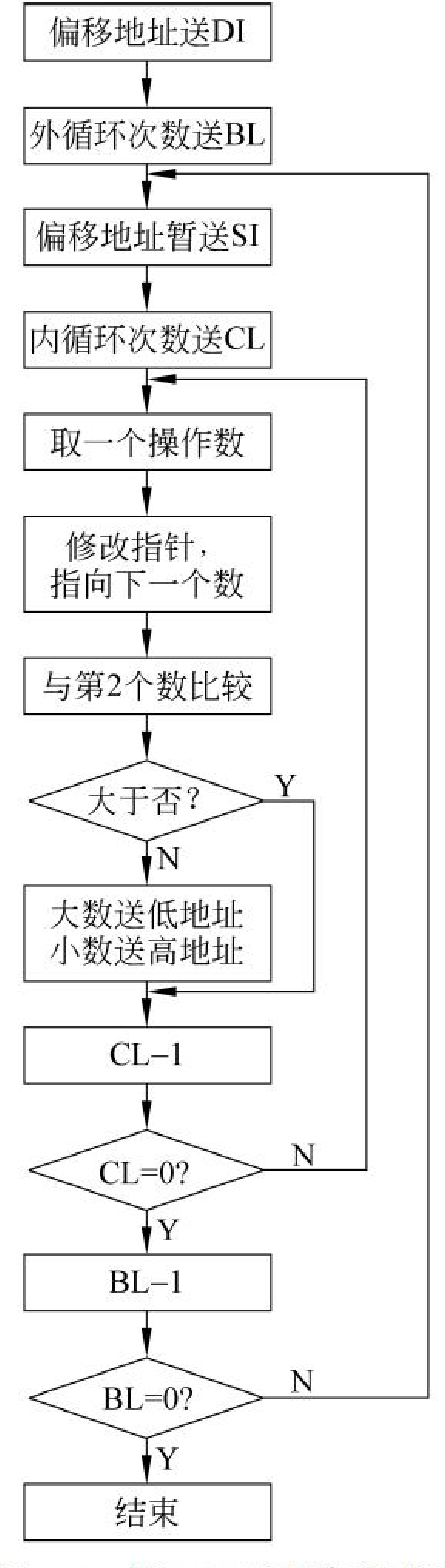


图 4-7 例 4-20 的程序流程图

4.4.5 子程序设计

子程序(或过程)是程序的一部分,是完成特定功能的程序段,它能够在程序中的任何地方被调用。在使用子程序时应注意以下 3 点。

(1) 参数的传递。在子程序调用时,经常需要将一些参数传送给子程序,而子程序也常常需要在运行后将结果和状态等信息回送给调用程序。这种子程序和调用程序之间的信息传送就称为参数传递。参数的传递可通过寄存器、变量、地址表、堆栈等方式进行。

(2) 相应寄存器的内容的保护。由于 CPU 的寄存器数量有限,子程序要用到的一些寄存器常在调用程序中也要用到。为防止破坏调用程序中寄存器的内容,需在子程序入口处将所用到的寄存器内容压入堆栈保存。

(3) 子程序还可调用别的子程序,称为子程序的嵌套。在多个子程序嵌套时,需要考虑堆栈空间的大小是否足以保存断点及相关寄存器参数。

前面已介绍过,与子程序调用有关的 CPU 指令有 CALL 和 RET;伪指令有 PROC 和 ENDP。

【例 4-21】 从一个字符串中删去一个字符。

题目分析:

这里,我们利用堆栈的方式来实现参数的传递,即在调用程序中将参数或参数地址保存在堆栈中,在子程序里再从堆栈中取出,从而实现参数的传送。

程序如下:

```
DATA    SEGMENT
STRING  DB 'Experience…'
LENG    DW $-STRING                  ;取字符串的长度
KEY     DB 'x'                       ;要从字符串中删去的字符
DATA    ENDS
  ;
CODE    SEGMENT
        ASSUME CS:CODE,DS:DATA,ES:DATA
MAIN    PROC FAR
START:  MOV AX,DATA
        MOV DS,AX
        MOV ES,AX
        LEA BX,STRING
        LEA CX,LENG
        PUSH BX
        PUSH CX                      ;将 STRING 和 LENG 的地址压栈
        MOV AL,KEY
```

```
            CALL DELCHAR            ;调用删除一个字符的子程序
            MOV AH,4CH
            INT 21H
MAIN        ENDP
DELCHAR     PROC
            PUSH BP                 ;保存 BP 内容
            MOV BP,SP               ;将 BP 指向当前栈顶
            PUSH SI
            PUSH DI
            CLD
            MOV SI,[BP+4]           ;得到 LENG 地址
            MOV CX,[SI]             ;取串长度
            MOV DI,[BP+6]           ;得到 STRING 地址
            REPNE SCASB             ;查找待删除的字符
            JNE DONE                ;若没有找到则退出
            MOV SI,[BP+4]
            DEC WORD PTR[SI]        ;串长度减 1
            MOV SI,DI
            DEC DI
            REP MOVSB               ;被删除字符后的字符依次向前移位
DONE:       POP DI                  ;恢复寄存器内容
            POP SI
            POP BP
            RET                     ;返回
DELCHAR     ENDP
     CODE ENDS
            END START
```

程序执行中堆栈最满时的状态如图 4-8 所示。

【例 4-22】 设一字符串长度不超过 255 个字符，试确定该字符串的长度并显示长度值。

题目分析：

字符串的长度不同于整数，系统并不规定为一个定值，所以在对字符串操作时常需要确定其长度。字符串通常以回车符“CR”或美元符“$”结尾。要确定一个字符串的长度可通过搜索字符串的结束标志来实现，即统计搜索次数直到找到结束符为止。若找不到结束符，则说明该字符串的长度超过了 255，程序应给出提示信息。

串长度可通过 DOS 功能调用显示。主程序和子程序的流程图如图 4-9 所示。

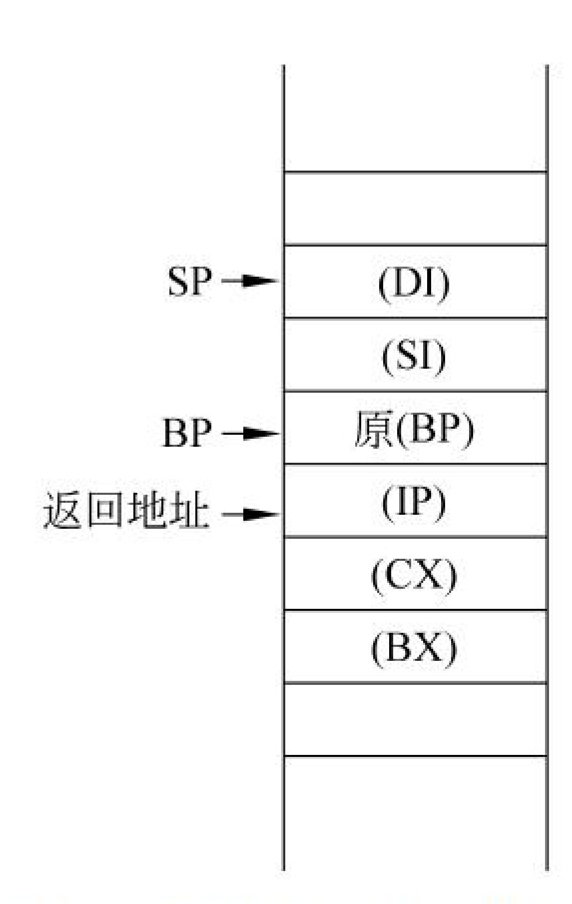


图 4-8　堆栈最满时的状态

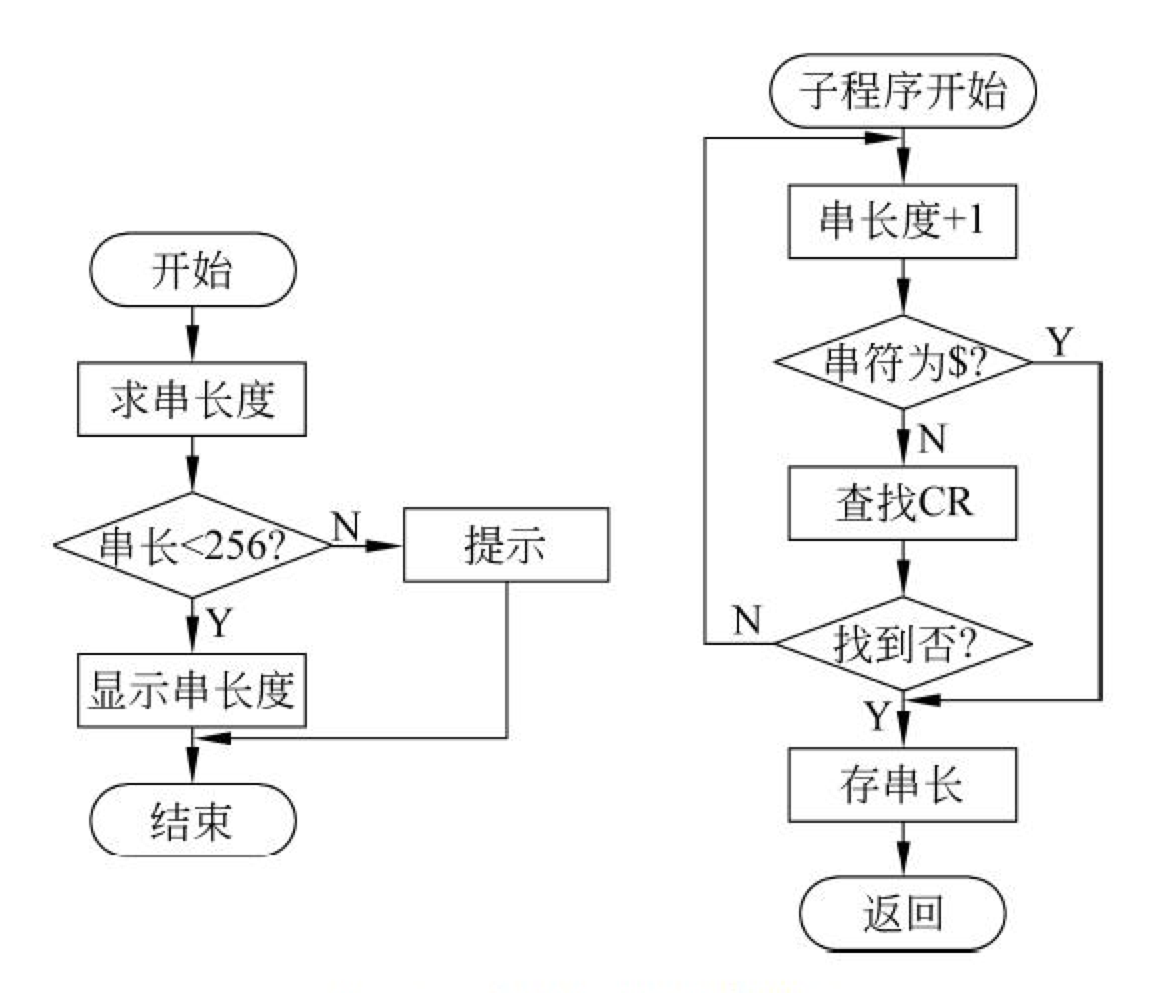


图 4-9　求串长度程序流程

程序代码如下：

```
DATA      SEGMENT
STRING    DB 'This is a string…',0DH
  LENG    DW ?
  CRR     DB 13                          ;定义回车符
MESSAGE   DB 'The string is too long! ',0DH,0AH,'$ '
DATA      ENDS
;
CODE    SEGMENT
        ASSUME CS:CODE,DS:DATA,ES:DATA
MAIN    PROC FAR
START:  MOV AX,DATA
        MOV DS,AX
        MOV ES,AX
        CALL STRLEN                      ;调用子程序,求字符串长度
        MOV DX,LENG
        CMP DX,100H
        JB NEXT1                         ;若 DX< 100H 则转 NEXT1
        LEA DX,MESSAGE                   ;若 DX≥100H 则显示提示信息
        MOV AH,9
        INT 21H
        JMP NEXT2
NEXT1:  MOV DH,DL                        ;串长度暂存 DH
        MOV CL,4
        SHR DL,CL                        ;取串长度高 4 位
        CMP DL,9
        JBE LP
        ADD DL,7
LP:     ADD DL,30H                       ;将串长度高 4 位转换为 ASCII 码
```

```
        MOV AH,2
        INT 21H                         ;显示串长度高 4位 ASCII码
        MOV DL,DH
        AND DL 0FH
        CMP DL,9
        JBE LP1
        ADD DL,7
LP1:    ADD DL,30H                      ;将串长度低 4位转换为 ASCII码
        MOV AH,2
        INT 21H                         ;显示串长度低 4位 ASCII码
        MOV DL,'H'
        MOV AH,2
        INT 21H
NEXT2:  MOV AH,4CH
        INT 21H
MAIN    ENDP
STRLEN PROC                             ;子程序
        LEA DI,STRING
        MOV CX,0FFFFH                   ;CX=-1
        MOV AL,CRR
        MOV AH,'$'
        CLD
AGAIN:  INC CX                          ;串长度+1
        CMP CX,100H
        JAE DONE                        ;串长度超过 255则结束
        CMP [DI],AH
        JE DONE                         ;遇到'$'则结束
        SCASB                           ;搜索回车符
        JNE AGAIN                       ;没找到则返回继续执行
DONE:   MOV LENG,CX
        RET
STRLEN ENDP
CODE    ENDS
        END START
```

4.4.6 常用程序设计举例

下面介绍一些常见的汇编语言程序设计的实例,供读者阅读。

【例 4-23】 把用 ASCII 码形式表示的数转换为二进制码。ASCII 码存放在以 MASC 为首地址的内存单元中,转换结果放 MBIN。

题目分析:

(1) 一般来讲,从键盘上输入的数都是以 ASCII 码的形式存放在内存中的,另外,数据区中以字符形式定义的数(用单引号括起来的数),在内存中也是以其对应的 ASCII 码

存放的；

(2) 对十六进制数来讲，0～9 的 ASCII 码分别为 30H～39H，对这 10 个数的转换，减去 30H，就得到对应的二进制值，而 A～F 的 ASCII 码分别为 41H～46H，故要减去 37H；

(3) 若取的数不在 0～FH 范围，则出错。

```
DATA      SEGMENT
MASC      DB '2','6','A','1'                  ;要转换的 ASCII 码
MBIN      DB 2 DUP(?)
DATA      ENDS
CODE      SEGMENT
          ASSUME CS:CODE,DS:DATA
BEGIN: MOV   AX,DATA
       MOV   DS,AX
       MOV   CL,4                            ;循环次数送 CL
       MOV   CH,CL                           ;保存循环次数
       LEA   SI,MASC                         ;ASCII 码单元首址送 SI
       CLD                                   ;按地址增量方向
       XOR   AX,AX                           ;中间结果清零
       XOR   DX,DX
NEXT1: LODSB                                 ;装入一个 ASCII 码到 AL
       AND   AL,7FH                          ;得到 7 位 ASCII 码
       CMP   AL,'0'
       JB    ERROR                           ;若 AL≤0 则转 ERROR
       CMP   AL,'9'
       JA    NEXT2                           ;若 AL≥9 则转 NEXT2
       SUB   AL,30H                          ;将 0～9 的数字转换为相应的二进制数
       JMP   SHORT NEXT3
NEXT2: CMP   AL,'A'
       JB    ERROR                           ;若 AL<'A',则转 ERROR
       CMP   AL,'F'
       JA    ERROR                           ;若 AL>'F',则转 ERROR
       SUB   AL,37H                          ;将 A~F 的数字转换为对应的二进制数
NEXT3: OR    DL,AL                           ;一个数的转换结果送 DL
       ROR   DX,CL                           ;整个转换的结果在 DX 中依次存放
ERROR: DEC   CH
       JNZ   NEXT1                           ;未转换完则转 NEXT1
       MOV   WORD PTR MBIN,DX                ;最后结果送 MBIN
       MOV   AH,4CH                          ;返回 DOS
       INT   21H
       CODE ENDS
             END    BEGIN
```

【例 4-24】 把存放在 BUFF 中的 16 位二进制数转换为 ASCII 码表示的等值数字字符串。例如，FFFFH 应转换成等值的数字字符串“65535”。

题目分析：

将一个二进制数转换为对应的ASCII码，可采用除10取余的方法。其基本思路为：任何一个用十六进制表示的二进制数，其除以10后的余数即是它对应十进制数的最低位，且一定在0～9之间。如“1234H”除以10，余数为4，用得到的余数加上30H，就得到了最低位对应的ASCII码。

16位二进制数能够表示的最大数字字符为“65535”。所以，最多除5次就可完成该二进制数的转换。

程序如下：

```
DATA   SEGMENT
BUFF   DW 4FB6H                      ;要转换的数
ASCC   DB 5 DUP(?)                   ;ASCII码结果存放单元
DATA   ENDS
CODE   SEGMENT
       ASSUME CS:CODE,DS:DATA
START: MOV  AX,DATA
       MOV  DS,AX
       MOV  CX,5                     ;最多不超过5位十进制数(65535)
       LEA  DI,ASCC                  ;DI指向结果存放单元
       XOR  DX,DX
       MOV  AX,BUFF                  ;取要转换的二进制数
       MOV  BX,0AH
AGAIN: DIV  BX                       ;用除10取余的方法转换
       ADD  DL,30H                   ;将余数转换成ASCII码
       MOV  [DI],DL                  ;保存当前位的结果
       INC  DI                       ;指向下一个位保存单元
       AND  AX,AX                    ;判断商是否为0(即转换是否结束)
       JZ   STO                      ;若结束,则退出
       MOV  DL,0
       LOOP AGAIN                    ;否则循环继续
STO:   MOV  AH,4CH
       INT  21H                      ;返回DOS
CODE   ENDS
       END START
```

【例4-25】 两个多字节二进制数求和程序。

题目分析：

由于8088/8086 CPU的内部寄存器均为16位。所以，在进行两个多字节数的求和运算时，一次只能完成一个字节或一个字的相加。低位字节（或字）相加的和可能会产生进位，那么在高位字节（或字）相加时则必须考虑该进位，否则就会使结果出错。因此，在多字节数求和运算中，要使用ADC指令，而不能使用ADD指令。

程序如下：

```
DATA   SEGMENT
BUFF1  DB 4FH,0B6H,7CH,34H, 56H,1FH              ;数1
```

```
BUFF2  DB 13H, 24H,57H,68H,0FDH,9AH                ;数 2
SUM    DB 6 DUP(?)                                 ;和
CONT   DB 3                                        ;数的字长为 3
DATA   ENDS
  ;
CODE   SEGMENT
       ASSUME CS:CODE,DS:DATA
START: MOV  AX,DATA
       MOV  DS,AX
       MOV  SI,OFFSET BUFF1                    ;SI 指向数 1
       MOV  DI,OFFSET BUFF2                    ;DI 指向数 2
       MOV  BX,OFFSET SUM                      ;BX 指向存放和的单元
       XOR  CX,CX                              ;使 CX=0,并同时使 CF←0
       MOV  CL,CONT                            ;共 3个字,要做 3次加法
GOON:  MOV  AX,[SI]                            ;取数 1的一个字
       ADC  AX,[DI]                            ;加上数 2的相应字
       PUSHF                                   ;保护状态标志
       ADD  SI,2                               ;修改指针
       ADD  DI,2
       MOV  [BX],AX                            ;存本次加的结果
       ADD  BX,2
       POPF                                    ;恢复状态标志
       LOOP GOON                               ;未加完,则循环
       ADC  BYTE PTR[BX],0                     ;保存最高位可能存在的进位
       MOV  AH,4CH
       INT  21H                                ;返回 DOS
       CODE ENDS
       END   START
```

【例 4-26】 从键盘上输入一个字符串,并在内存中已有的一张表中查找该字符串,若找到则在屏幕上显示“OK!”,否则显示“NO!”,若输入字符长度大于表长度,则显示“Wrong!”。

题目分析:

(1) 在查找前,首先判断输入的字符串的长度是否大于表的长度,若大于则表示输入的字符串太长,显示“Wrong!”,否则就进行比较;

(2) 先在表中查找字符串的第一个字符,若找到,再比较字符串的其他字符是否一致;

(3) 在屏幕上显示一个字符串可利用 DOS 功能调用中的 09 功能号,而从键盘上接收一个字符串可利用功能号为 0AH 的 DOS 调用。

程序如下:

```
DATA      SEGMENT
TABLE     DB 'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
STRING1   DB 'Please enter a string:',0DH,0AH,'$ '
STRING2   DB 'Wrong! ',0DH,0AH,'$ '
STRING3   DB 'OK! ','$ '
```

```
STRING4   DB 'NO! ','$'
BUFFER    DB 40,?,40 DUP(?)                    ;键盘输入缓冲区
TAB_LEN   EQU 26
DATA      ENDS
  ;
STACK   SEGMENT
        DB 100 DUP(?)
STACK   ENDS
  ;
CODE  SEGMENT
      ASSUME CS:CODE,DS:DATA,ES:DATA,SS:STACK
START: MOV    AX,DATA
       MOV    DS,AX
       MOV    ES,AX
       LEA    DX,STRING1                      ;显示"Please enter a string:"
       MOV    AH,09H
       INT    21H
       LEA    DX,BUFFER                       ;从键盘读字符串
       MOV    AH,0AH
       INT    21H
       MOV    SI,DX                           ;串首地址送 SI
       INC    SI
       MOV    BL,[SI]
       MOV    BH,0                            ;串长度送 BX
       INC    SI                              ;串首地址送 SI
       LEA    DI,TABLE                        ;表首地址送 DI
       MOV    CX,TAB_LEN                      ;表长度送 CX
       CMP    CX,BX                           ;表长≥串长?
       JNC    GOON                            ;是则转 GOON
       LEA    DX,STRING2                      ;否则显示"Wrong!"
       JMP    EXIT
GOON:  CLD                                    ;按增地址方向进行比较
       MOV    AL,[SI]
SCAN:  REPNZ  SCASB                           ;在表中搜索第一个字符
       JZ     MATCH                           ;找到则转 MATCH
ERROR: LEA    DX,STRING4                      ;没有找到,显示"NO!"
       JMP    EXIT
MATCH: INC    CX
       CMP    CX,BX                           ;剩余表长≥串长?
       JC     ERROR                           ;不大于,显示"NO!"
       PUSH   CX                              ;保存循环变量
       PUSH   SI
       PUSH   DI
       MOV    CX,BX
       DEC    DI
       REPZ   CMPSB                           ;比较串中其余字符
```

```
        POP    DI                                    ;恢复循环变量
        POP    SI
        POP    CX
        JZ     FOUND                                 ;若找到字符串,转 FOUND
        JCXZ   ERROR                                 ;未找到字符串,且全表搜索完,转 ERROR
        JMP    SCAN                                  ;全表未搜索完,转 SCAN
FOUND:  DEC DI                                       ;找到的字符串偏移地址送 DI
        LEA    DX,STRING3                            ;显示"OK!"
EXIT:   MOV    AH,09H
        INT    21H
        MOV    AH,4CH                                ;返回 DOS
        INT    21H
CODE           ENDS
        END    START
```

【例 4-27】 在分辨率为 640×480、16 色的屏幕上绘制一个周期的正弦波。

题目分析：

(1) 正弦波一个周期的角度值范围为 0°～360°,函数值范围为－1～1。要使曲线居于屏幕正中,必须要调整水平和垂直方向的坐标值。

(2) 在给定 0°～90°的函数值情况下,绘制正弦波曲线时须先知道角度所在的象限。

① 若角度在第Ⅰ象限,函数值为正。此时可直接查表取函数值。

② 若角度在第Ⅱ象限,函数值为正。可利用 $\sin(X)=\sin(180°-X)$,将角度转换到第Ⅰ象限后再查表取函数值。

③ 若角度在第Ⅲ或第Ⅳ象限,函数值为负。先将 $X-180°$转换到第Ⅰ或第Ⅱ象限,再按前述处理,并把结果取负值。

(3) 为简化程序设计,可在绘图前先计算出曲线各点的坐标值并列成表格,这样在画图时只需访问这个表格就可以了。设正弦波图形范围为 360×400,表格 SINE 中为从 0°～90°的放大 200 倍的已取整的正弦值。

程序代码如下：

```
SETSCREEN MACRO                ;设置屏幕分辨率为 640×480,16 色图形方式
          MOV AH,0
          MOV AL,12H
          INT 10H
ENDM
WRITEDOT MACRO                 ;画点宏定义
         MOV AH,0CH
         MOV AL,02H            ;像素颜色代码
         MOV CX,ANGLE          ;像素点对应的列号送 CX
         ADD CX,140            ;X方向屏幕中心=(640-360)/2
         MOV DX,TEMP           ;像素点所在的行号送 DX
         INT 10H
ENDM
```

```
DATA SEGMENT
SINE DB 00,03,07,10,14,17,21,24,28,31,       ;定义坐标表格
        35,38,42,45,48,52,55,58,62,65,
        68,72,75,78,81,85,88,91,94,97,
        100,103,106,109,112,115,118,120,
        123,126,129,131,134,136,139,141,
        144,146,149,151,153,155,158,160,
        162,164,166,168,170,171,173,175,
        177,178,180,181,183,184,185,187,
        188,189,190,191,192,193,194,195,
        196,196,197,198,198,199,199,199,
        200,200,200,200
ANGLE DW 0                         ;定义角度变量,初值为 0
TEMP DW 0                          ;定义点的正弦函数值变量,初值为 0
DATA ENDS
STACK SEGMENT
      DB 64 DUP(?)
STACK ENDS
CODE SEGMENT
     ASSUME CS:CODE,DS:DATA,SS:STACK
MAIN PROC FAR
START: PUSH DS                     ;保护参数
       PUSH AX
       PUSH BX
       MOV AX,DATA
       MOV DS,AX
       MOV AX,STACK
       MOV SS,AX

;查表确定正弦波函数值,逐点绘制正弦波
        SETSCREEN                  ;置屏幕为 640×480 的彩色图形方式
AGAIN: LEA BX,SINE                 ;表的偏移地址送 BX
       MOV AX,ANGLE                ;角度值送 AX
       CMP AX,180                  ;看是否大于 180°
       JLE QUAD1                   ;若不大于则角度在第Ⅰ或第Ⅱ象限
       SUB AX,180                  ;若大于则调整角度
QUAD1: CMP AX,90                   ;大于 90°否?
       JLE QUAD2                   ;若不大于则角度在第Ⅰ象限
       NEG AX                      ;否则角度在第Ⅱ象限
       ADD AX,180                  ;调整角度(180-ANGLE)
QUAD2: ADD BX,AX                   ;形成查表偏移量
       MOV AL,SINE[BX]             ;将函数值送 AL
       PUSH AX
       MOV AH,0
```

```
        CMP ANGLE,180          ;判断函数值是否大于 180°
        JGE BIGDIS             ;若大于则转 BIGDIS
        NEG AL                 ;否则在第Ⅰ或第Ⅱ象限
        ADD AL,240             ;调整显示点的纵坐标为(240-AL)
        JMP READY
BIGDIS:ADD AX,240              ;调整显示点的纵坐标为(240+AL)
READY: MOV TEMP,AX             ;保存到 TEMP
        POP AX
        WRITEDOT               ;调用画点宏操作
        ADD ANGLE,1            ;角度值+1
        CMP ANGLE,360          ;超过 360°吗?
        JLE AGAIN              ;不超过则继续画
        MOV AH,07              ;若有键按下则继续执行,否则等待按键输入
        INT 21H
        MOV AH,0               ;设置屏幕参数
        MOV AL,3               ;设置 80×25 彩色文本方式
        INT 10H
        POP BX                 ;恢复参数
        POP AX
        POP DS
        RET                    ;返回
MAIN ENDP
CODE ENDS
     END START
```

习　　题

4.1　分别用 DB、DW、DD 伪指令写出在 DATA 开始的连续 8 个单元中依次存放数据 11H、22H、33H、44H、55H、66H、77H、88H 的数据定义语句。

4.2　若程序的数据段定义如下,写出各指令语句独立执行后的结果。

```
DSEG SEGMENT
DATA1 DB 10H,20H,30H
DATA2 DW 10 DUP(?)
STRING DB '123'
DSEG ENDS
```

(1) MOV AL,DATA1

(2) MOV BX,OFFSET DATA2

(3) LEA SI,STRING
　　MOV DI,WORD PTR DATA1
　　ADD DI,SI

4.3　试编写求两个无符号双字长数之和的程序。两数分别在 MEM1 和 MEM2 单元中,

和放在 SUM 单元中。

4.4 试编写程序，测试 AL 寄存器的第 4 位(D_4)是否为 0。

4.5 试编写程序，将 BUFFER 中的一个 8 位二进制数的高 4 位和低 4 位分别转换为 ASCII 码，并按位数高低顺序存放在 ANSWER 开始的内存单元中。

4.6 假设数据项定义如下：

```
DATA1 DB 'HELLO!GOOD MORNING!'
DATA2 DB 20 DUP(?)
```

用串操作指令编写程序段，使其分别完成以下功能。

(1) 从左到右将 DATA1 中的字符串传送到 DATA2 中。

(2) 传送完后，比较 DATA1 和 DATA2 中的内容是否相同。

(3) 把 DATA1 中的第 3 和第 4 个字节装入 AX。

(4) 将 AX 的内容存入 DATA2+5 开始的字节单元中。

4.7 执行下列指令后，AX 寄存器中的内容是多少？

```
TABLE DW 10,20,30,40,50
ENTRY DW 3
⋮
MOV BX,OFFSET TABLE
ADD BX,ENTRY
MOV AX,[BX]
```

4.8 编写程序段，将 STRING1 中的最后 20 个字符移到 STRING2 中(顺序不变)。

4.9 假设一个 48 位数存放在 DX:AX:BX 中，试编写程序段，将该 48 位数乘以 2。

4.10 试编写程序，比较 AX、BX、CX 中带符号数的大小，并将最大的数放在 AX 中。

4.11 若接口 03F8H 的第 1 位(D_1)和第 3 位(D_3)同时为 1，表示接口 03FBH 有准备好的 8 位数据，当 CPU 将数据取走后，D_1 和 D_3 就不再同时为 1 了。仅当又有数据准备好时才再同时为 1。

试编写程序，从上述接口读入 200 字节的数据，并顺序放在 DATA 开始的地址中。

4.12 画图说明下列语句分配的存储空间及初始化的数据值。

(1) DATA1 DB ‘BYTE’,12,12H,2 DUP(0,?,3)

(2) DATA2 DW 4 DUP(0,1,2),?,−5,256H

4.13 请用子程序结构编写如下程序：从键盘输入一个二位十进制的月份数(01～12)，然后显示出相应的英文缩写名。

4.14 给出下列等值语句：

```
ALPHA EQU 100
BETA EQU 25
GRAMM EQU 4
```

试求下列表达式的值。

(1) ALPHA×100+BETA；

(2) (ALPHA+4)×BETA−2；

(3) (BETA/3)MOD 5；

(4) GRAMM OR 3

4.15 画图说明以下数据段在存储器中的存放形式。

```
DATA SEGMENT
DATA1 DB 10H,34H,07H,09H
DATA2 DW 2 DUP(42H)
DATA3 DB 'HELLO!'
DATA4 EQU 12
DATA5 DD 0ABCDH
DATA ENDS
```

4.16 阅读下面的程序段，试说明它实现的功能。

```
DATA SEGMENT
DATA1 DB 'ABCDEFG'
DATA ENDS
CODE SEGMENT
    ASSUME CS:CODE,DS:DATA
AAAA: MOV AX,DATA
      MOV DS,AX
      MOV BX,OFFSET DATA1
      MOV CX,7
NEXT: MOV AH,2
      MOV AL,[BX]
      XCHG AL,DL
      INC BX
      INT 21H
      LOOP NEXT
      MOV AH,4CH
      INT 21H
CODE  ENDS
      END AAAA
```

4.17 编写程序，实现将 BUFFER 开始的 100 个字节的内存区域初始化成 55H、0AAH、55H、0AAH、…、55H、0AAH。

4.18 有 16 个字节，编程将其中第 2、5、9、14、15 个字节内容加 3，其余字节内容乘 2(假定运算不会溢出)。

4.19 编写计算斐波那契数列前 20 个值的程序。斐波那契数列的定义如下：

$$\begin{cases} F(0)=0 \\ F(1)=1 \\ F(n)=F(n-1)+F(n-2), \quad n\geqslant 2 \end{cases}$$

4.20 试编写将键盘输入的 0～F 转换为二进制数的程序。



第5章 存储器系统

引言：

每个基于微处理器的系统都有存储器。几乎所有的系统都包含两类主要的存储器：只读存储器(ROM)和随机存取存储器(RAM)。ROM 存放系统软件和永久性系统数据，而 RAM 则通常用于存放临时数据和应用程序。在现代微机系统中，它们作为内存储器而成为主机系统的一个重要组成部件，其自身或与磁盘存储器一起构成存储器系统，在整个微机系统中占据着越来越重要的位置。

本章在介绍存储器系统的基本概念和构成的基础上，主要介绍如何将两类半导体存储器芯片与 CPU 进行连接以及如何利用已有的存储器芯片构成所需要的内存空间。

教学目的：

(1) 了解存储器系统的基本概念及不同类型半导体存储器的特点；

(2) 熟练掌握典型半导体存储器芯片与系统的连接；

(3) 掌握存储器扩展技术；

(4) 了解高速缓冲存储器的概念及其一般工作原理。

5.1 概　　述

从程序员的角度看，计算机必须把相应的程序和数据装入存储器才能开始运行。存储器是计算机系统的记忆设备，用于存放计算机要执行的指令、处理的数据、运算结果以及各种需要保存的信息，是计算机中不可缺少的一个重要组成部分。从记忆信息的角度讲，计算机中的存储器就相当于人的大脑。

由第 2 章的内容可知，在程序执行过程中，中央处理器从存储器中取得指令。运算指令中需要的数据也要通过访问存储器指令从存储器中取得。而运算结果在程序结束前必须全部写入存储器中。各种输入输出设备也直接与存储器交换数据。因此，存储器是计算机运行过程中信息存储交换的中心设备，从这个意义上说，现代计算机系统是以存储器为中心的。

存储器有两种基本操作——读和写。读操作是指从存储器中读出信息，不破坏存储单元中原有的内容；写操作是指把信息写入(存入)存储器，新写入的数据将覆盖原有的内容。

5.1.1 存储器系统的一般概念

存储器系统与存储器是两个不同的概念。在现代计算机中通常有多种用途的存储器件，如内存、高速缓存(Cache)、磁盘、可移动硬盘、磁带、光盘等。它们的工作速度、存储容量、单位容量价格、工作方式以及制造材料等各方面都不尽相同。存储器系统的概念是：将两个或两个以上速度、容量和价格各不相同的存储器用软件、硬件或软硬件相结合的方法连接起来，成为一个系统。这个系统从程序员的角度看，是一个存储器整体。所构成的存储器系统的速度接近于其中速度最快的那个存储器，存储容量与存储容量最大的那个存储器相等或接近，单位容量的价格接近最便宜的那个存储器。对于一个计算机系统，存储器系统的优劣，特别是它的存取速度和存储容量关系着整个计算机系统的优劣。

1. 微机中的存储器系统

现代微机系统中通常有两种存储系统：①由 Cache 和主存储器构成的 Cache 存储系统，如图 5-1 所示；②由主存储器和磁盘构成的虚拟存储系统，如图 5-2 所示。两种存储系统的作用各不相同，前者的主要目标是提高存取速度，而后者的主要目标是增加存储容量。



图 5-1 Cache 存储系统　　图 5-2 虚拟存储系统

(1) Cache 存储系统的管理全部由硬件实现，无须系统程序员干预，即它对软件开发设计人员是透明的(一个实际存在的部件看起来好像不存在，称为“透明”)。

Cache 一般由高速静态存储器(SRAM)组成，存取周期为零点几个纳秒，存储容量在几十 KB 至几十 MB 之间，价格较高。主存一般由动态存储器(DRAM)组成，存储周期一般为几纳秒到几十纳秒，存储容量可达几百 MB 到几 GB，价格比 Cache 相对便宜得多。这两种存储器组成了 Cache 存储系统。

Cache 存储系统在设计上，力争在一定的时间区间内，CPU 需要的指令和数据都能在 Cache 中访问到，因此，这个存储系统的存取周期与 Cache 非常接近。由于 Cache 中的数据和地址都是主存相应内容和地址的映像，它们之间的地址变换和映像都由硬件系统管理，对程序员来说“看不到”Cache，所以在编程时，只需要对主存储器编址。因此，Cache 存储器系统的容量就是主存储器的容量。由于 Cache 的容量相比主存的容量要小得多(通常为 1∶128)，故整个 Cache 存储器系统每单位的平均价格与主存储器很接近。

Cache 中存放 CPU 最近一直在使用的指令和数据。当 Cache 装满后，可将长期不用的数据删除，以提高 Cache 的使用效率。

(2) 虚拟存储系统由主存储器与磁盘存储器构成。在早期的微机中，磁盘等外存储

器作为外部设备的一部分,仅用于长期保存信息。由于内存容量很小,程序员必须花费很大精力将大程序预先分成块,确定好这些程序块在外存设备中的位置和装入主存的地址,并且在运行中还要预先安排好各块如何和何时调入调出。

由于磁盘存储器不是能随机访问的存储器,即不能被一般指令直接访问,而主存储器的地址空间对使用者来讲又太小,因此,现代虚拟存储系统在操作系统的支持下为用户另外设计了一个虚拟地址空间。它将主存和外存看做一个整体,用软硬件相结合的方法进行管理,使得程序员能够对主存、辅存统一编址,这样形成的一个很大的地址空间,称为虚拟地址空间。虚拟地址空间比实际主存储器的存储容量大得多,32 位微机可访问的编址空间为 4GB。虚拟存储系统在构成原理上与 Cache 存储器系统类似,其访问速度接近主存的速度。由于磁盘存储器每单位容量的价格比主存储器便宜很多,因此,整个存储系统的每单位容量的平均价格接近磁盘存储器。

虚拟地址空间既不是主存地址空间,也不是磁盘存储器的地址空间,它是为使用者设计的一个逻辑地址空间。它远大于主存储器的实际地址空间,在软硬件系统的支持下,可以采用与主存储器同样的随机访问方式。

2. 存储系统的主要性能指标

衡量存储器系统的性能主要有 3 个参数:存储容量 S 、存取时间 T 和单位容量价格 C,组成存储系统的每个存储器也有 3 个同样的参数,通过分析这些参数之间的关系,可以评测一个存储系统。

1) 存储容量 S

设有两种存储器 M_1 和 M_2,它们组成一个存储系统。两种存储器的容量、速度和价格分别为 S_1、T_1、C_1 和 S_2、T_2、C_2,存储系统的容量、速度和价格分别为 S、T、C。

对于 Cache 存储系统,由系统程序员看来,存储系统的容量接近主存储器的容量,故选择主存 M_2 进行编址,对 Cache 在内部采用相连访问方式管理。因此,系统程序员看到的是主存储器的地址空间,存储系统的容量就是主存储器的容量,$S=S_2$。

对于虚拟存储系统,它的地址空间比主存储器大得多。还应当说明的是,在一般计算机系统中,并不是整个磁盘存储器都作为虚拟存储系统使用。磁盘存储器的主要用途仍然是用来存放系统软件、应用软件和用户文件,只有在多任务多用户操作系统的交换区或交换文件才用来作虚拟存储系统。

2) 存取时间 T

存储器系统的存取时间与命中率 H 有关。命中率表示从速度较快的那个存储器中访问到数据的概率,一般用模拟试验的方法得到。在一组有代表性的程序执行中,分别统计对 M_1 的访问次数 N_1 和对 M_2 的访问次数 N_2,然后代入关系式(5.1)

$$H = \frac{N_1}{N_1 + N_2} \tag{5.1}$$

整个存储器系统的存取时间可以用 M_1 和 M_2 两个存储器的存取时间 T_1、T_2 和命中率 H 来表示

$$T = H \cdot T_1 + (1 - H) \cdot T_2 \tag{5.2}$$

当命中率 $H \to 1$ 时，$T \to T_1$，即存储器系统的速度接近于较快的 M_1 存储器的存取周期 T_1。

设存储器系统的访问效率为

$$e = \frac{T_1}{T} \tag{5.3}$$

存储器系统的速度与相对较快的那个存储器的速度越接近，访问效率就越高。把式(5.2)代入式(5.3)得到

$$e = \frac{T_1}{H \cdot T_1 + (1-H) \cdot T_2} = \frac{1}{H + (1-H) \cdot \frac{T_2}{T_1}} = f\left(H, \frac{T_2}{T_1}\right) \tag{5.4}$$

可以看出，存储系统的访问效率主要与命中率和构成存储系统的两级存储器的速度之比有关。因此，如果要使存储系统的访问效率提高有两条途径：①提高命中率 H；②使构成存储系统的两级存储器的速度之比不要太大。

对于虚拟存储系统，由于磁盘的存取操作还要依赖机械运动，两级存储器的速度相差悬殊，主存储器的存取速度为纳秒级，硬盘存取速度为毫秒级。若要使访问效率 e 比较高(如 $e=0.9$)，需要极高的命中率 H，如果 $T_2/T_1>10^5$，则依式(5.4)计算 H 约为 0.999999，如何使用现有技术达到高命中率呢？

因为磁盘在物理上是以块为单位(每块 512 个字节)访问的，所以，虽然磁盘存储器的寻址定位时间很长，但当磁盘找到要访问的连续的数据块之后，数据的传输速率还是相当高的。因此，当不命中时，通过操作系统的功能调用，把将要使用的一大批程序和数据都调入主存储器，使得在以后的多次对虚拟存储系统的访问都能在主存储器命中。只要主存储器的容量比较大，能够一次装入比较多的程序和数据，这样，尽管两级存储器的速度差异悬殊，一次不命中需要花费较长的时间进行调度，然而由于命中率特别高，整个虚拟存储系统的访问效率还是很高的。

Cache 存储系统要缓冲 CPU 和主存之间的速度差异，目前 CPU 与主存储器的速度相差两个数量级，如果要求 $H \approx 0.999$，用一级 Cache 是做不到的。通常采用两级或三级 Cache，再加上 CPU 内部的一些缓冲存储器，像通用寄存器等来提高数据的重复利用率，使得每两级之间的速度比减小。另外，再采用预取技术以大幅度提高命中率：当不命中时，在数据从主存储器取出送往 CPU 的同时，把主存储器相邻几个单元中的数据(一个数据块)都取出来送入 Cache 中。CPU 以后再对 Cache 存储系统进行访问时，命中率就会提高。

3）单位容量的平均价格 C

整个存储器系统的单位容量平均价格可以用下式计算

$$C = \frac{C_1 \times S_1 + C_2 \times S_2}{S_1 + S_2} \tag{5.5}$$

当 S_2 大大超过 S_1 时，$C \approx C_2$。这时，整个存储系统的单位容量价格 C 接近于比较便宜的 M_2 存储器的单位容量价格 C_2。但是 S_2 和 S_1 的差距应在一个合理的范围内，如果差距太大，存储器要达到较高的性能，调度安排将会很困难。

5.1.2 半导体存储器及其分类

计算机的存储器从体系结构的观点来划分,可根据其是设在主机内还是主机外分为内部存储器和外部存储器两大类。内部存储器主要由半导体材料制成,也称半导体存储器;外部存储器由磁性材料或复合材料制造,包括硬磁盘、软盘、可移动硬盘、磁带、CD-ROM 等。这两种类型的存储器在性能上主要有以下几个方面的特点。

(1) 内存(或称主存)是计算机主机的组成部分之一,用来存储当前运行所需要的程序和数据,CPU 可以直接访问并与其交换信息;而外存属于外部设备,CPU 不能对它直接访问,必须通过专门的接口才能实现与 CPU 的信息交换。

(2) 内存的容量小、存取速度快、价格相对较高;外存储器的容量大、价格低,但速度慢。

(3) 内存是数据的"临时住所",主要用于存放程序运行时所需的信息。当程序运行结束或关机后,除少量信息(如 BIOS 等)外,其他信息都会立即消失。而外存储器则用于大容量、永久性数据的保存。

限于篇幅,第 5 章仅通过一些典型半导体存储器芯片介绍内部存储器的工作原理。对于外部存储器技术,读者可参阅其他相关书籍。

1. 存储元

半导体存储器由一些能够表示二进制"0"和"1"的状态的物理器件组成,这些器件本身具有记忆功能,如电容、双稳态电路等。将这些具有记忆功能的物理器件叫做存储元(如一个电容就是一个存储元)。每个存储元可以保存一位二进制信息。若干个存储元就构成了一个存储单元。在微机系统中,一个存储单元通常存放 8 位二进制码(1B),即一个存储单元由 8 个存储元构成,许多存储单元组织在一起就构成了存储器。

我们把存储器中存储单元的总数称为存储器的存储容量。显然,存储容量越大,能够存放的信息就越多,计算机的信息处理能力也就越强。

2. 半导体存储器的分类

半导体存储器按照工作方式的不同,可分为随机存取存储器(Random Access Memory,RAM)和只读存储器(Read Only Memory,ROM)。

1) 随机存取存储器 RAM

RAM 的主要特点是可以随机进行读写操作,但掉电后信息会丢失,是目前微机中主内存的主要构成部件。根据制造工艺的不同,RAM 可以分为双极型半导体 RAM 和金属氧化物半导体(MOS)RAM。双极型 RAM 的主要优点是存取时间短,通常为几纳秒到几十纳秒(ns)。与 MOS 型 RAM 相比,其集成度低、功耗大,而且价格也较高。因此,双极型 RAM 主要用于要求存取时间非常短的特殊应用场合(如高速缓冲存储器 Cache)。

用 MOS 器件构成的 RAM 又可分为静态读写存储器(SRAM)和动态读写存储器(DRAM)。SRAM 的存储元由双稳态触发器构成。双稳态触发器有两个稳定状态,可用

来存储一位二进制信息。只要不掉电，其存储的信息可以始终稳定地存在，故称其为静态RAM。SRAM的主要特点是存取时间短(几十到几百纳秒)、外部电路简单、便于使用。常见的SRAM芯片容量为1～64KB。

DRAM的存储元以电容来存储信息，电路简单。但电容总有漏电存在，时间长了存放的信息就会丢失或出现错误。因此需要对这些电容定时充电，这个过程称为刷新，即定时地将存储单元中的内容读出再写入。由于需要刷新，所以这种RAM称为动态RAM。DRAM的存取速度一般较SRAM的存取速度低。其最大的特点是集成度非常高，目前DRAM芯片的容量已达几百MB。其他的优点还有功耗低、价格比较便宜。

由于用MOS工艺制造的RAM集成度高，存取速度能满足各种类型微型机的要求，而且其价格也比较便宜，因此，现在微型计算机中的内存主要由MOS型DRAM组成。

2) 只读存储器ROM

只读存储器包括掩膜ROM、PROM、EPROM、E^2PROM等几种类型。ROM的主要特点是掉电后不会丢失所存储的内容，可随机进行读操作，但不能写入或只能有条件编程写入，常用于存放一些相对不变的数据(如BIOS等)。

(1) 掩膜式只读存储器(ROM)。掩膜式ROM是芯片制造厂根据要存储的信息，对芯片图形(掩膜)通过二次光刻生产出来的，故称为掩膜ROM。其存储的内容固化在芯片内，用户可以读出，但不能改变。这种芯片存储的信息稳定、成本最低，适用于存放一些可批量生产的固定不变的程序或数据。

(2) 可编程ROM(Programable ROM，PROM)。如果用户要根据自己的需要来确定ROM中的存储内容，则可使用PROM。PROM允许用户对其进行一次编程－写入数据或程序。一旦编程之后，信息就永久性地固定下来。用户可以读出其内容，但再也无法改变它的内容。

(3) 可读写ROM。上述两种芯片存放的信息只能读出而无法修改，这给许多方面的应用带来不便。由此又出现了可读写的ROM芯片。这类芯片允许用户通过一定的方式多次写入数据或程序，也可修改和擦除其中所存储的内容，且写入的信息不会因为掉电而丢失。由于这些特性，可读写ROM芯片在系统开发、科研等领域得到了广泛的应用。

可读写ROM芯片因其擦除的方式不同又可分为两类：通过紫外线照射(约20分钟)实现擦除的称为EPROM(Erasable Programable ROM)；另外一种通过电信号(通常是加上一定的电压)进行擦除的ROM称为EEPROM(Electrically Erasable PROM)(或E^2PROM)。这两种芯片的内容在擦除后仍可重新编程写入新的内容，擦除和写入都可以多次进行。但有一点要注意，尽管EPROM和EEPROM芯片都是既可读出也可写入和擦除，但它们和RAM还是有本质区别的。首先它们不能够像RAM芯片那样随机快速地写入和修改，它们的写入需要一定的条件(这一点将在后面详细介绍)；另外，RAM中的内容在掉电之后会丢失，而EPROM(EEPROM)则不会，其上的内容一般可保存几十年。

5.1.3 半导体存储器的主要技术指标

1. 存储容量

存储器芯片的存储容量用“存储单元个数×每存储单元的位数”来表示。例如，SRAM 芯片 6264 的容量为 8K×8b，即它有 8K 个单元（1K＝1024），每个单元存储 8 位（一个字节）二进制数据。DRAM 芯片 NMC41257 的容量为 256K×1b，即它有 256K 个单元，每个单元存储 1 位二进制数据。各半导体器件生产厂家为用户提供了许多种不同容量的存储器芯片，用户在构成计算机内存系统时，可以根据要求加以选用。当然，当计算机的内存确定后，选用容量大的芯片则可以少用几片，这样不仅使电路连接简单，而且功耗也可以降低。

2. 存取时间和存取周期

存取时间又称存储器访问时间，即启动一次存储器操作（读或写）到完成该操作所需要的时间。CPU 在读写存储器时，其读写时间必须大于存储器芯片的额定存取时间。如果不能满足这一点，微型机则无法正常工作。

存取周期是连续启动两次独立的存储器操作所需间隔的最小时间。若令存取时间为 t_A，存取周期为 T_C，则二者的关系为 $T_C \geqslant t_A$。

3. 可靠性

计算机要正确地运行，必然要求存储器系统具有很高的可靠性。内存发生的任何错误都会使计算机不能正常工作，而存储器的可靠性直接与构成它的芯片有关。目前所用的半导体存储器芯片的平均故障间隔时间（MTBF）为 $5\times10^6 \sim 1\times10^8$ 小时。

4. 功耗

使用功耗低的存储器芯片构成存储系统，不仅可以减少对电源容量的要求，而且还可以提高存储系统的可靠性。

5.2 随机存取存储器

随机存取存储器（RAM）主要用来存放当前运行的程序、各种输入输出数据、中间运算结果及堆栈等，其存储的内容既可随时读出，也可随时写入和修改，掉电后内容会全部丢失。5.2 节将从应用的角度出发，以几种常用的典型芯片为例，详细介绍两类 MOS 型随机存取存储器——SRAM（Static RAM）和 DRAM（Dynamic RAM）——的特点、外部特性以及它们的应用。

5.2.1 静态随机存取存储器

静态随机存取存储器(SRAM)的基本存储电路(即存储元)一般是由 6 个 MOS 管组成的双稳态电路(T_1 截止,T_2 导通为状态"1";T_2 截止,T_1 导通为状态"0"),如图 5-3 所示。

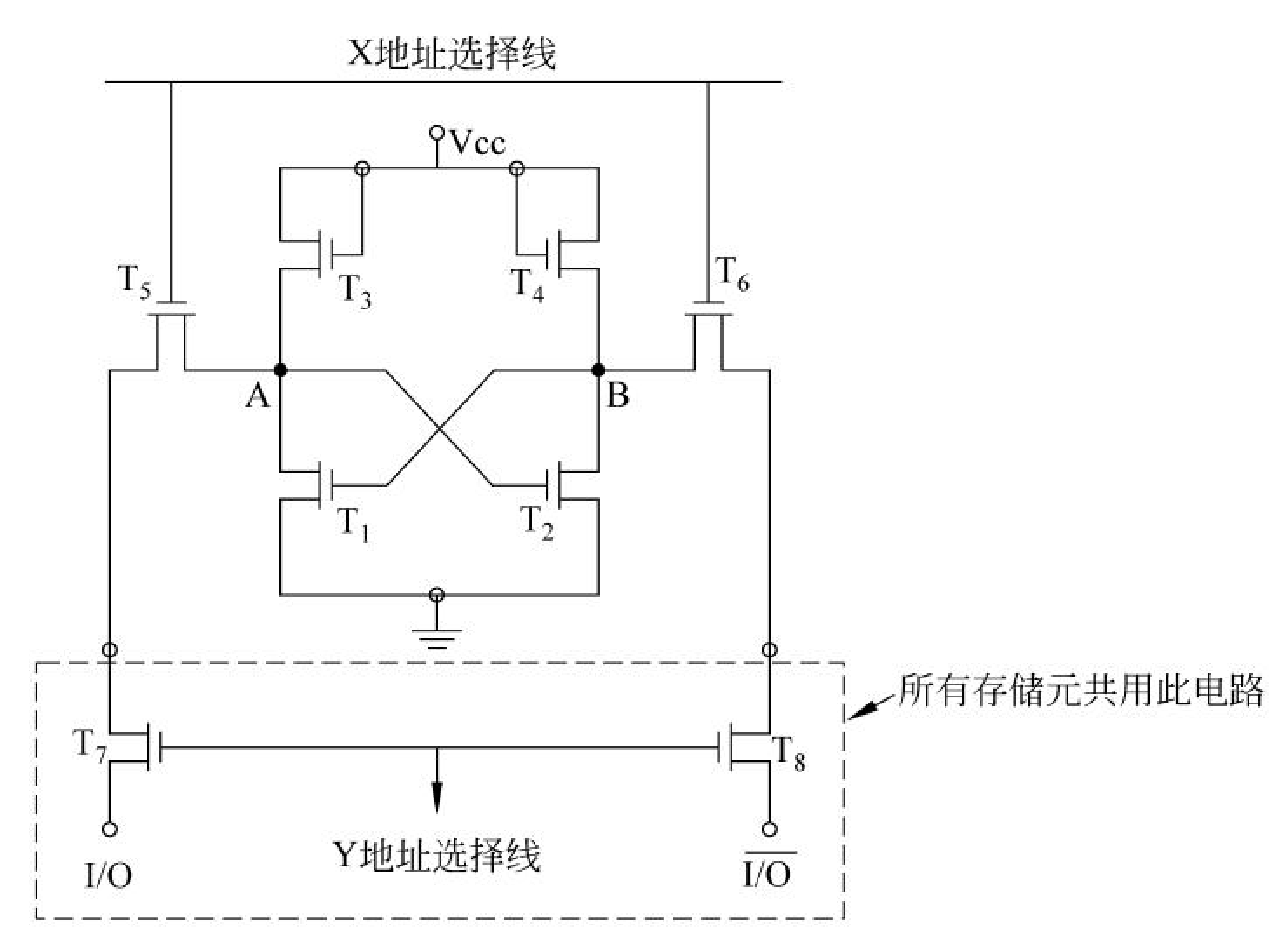


图 5-3 SRAM 的基本存储电路

图 5-3 中,T_3、T_4 是负载管,T_1、T_2 是工作管,T_5、T_6、T_7、T_8 是控制管,其中 T_7、T_8 为所有存储元共用。

在写操作时,若要写入"1",则 I/O=1,$\overline{I/O}$=0,X 地址选择线为高电平,使 T_5、T_6 导通,同时 Y 地址选择线也为高电平,使 T_7、T_8 导通,要写入的内容经 I/O 端和$\overline{I/O}$端进入,通过 T_7、T_8 和 T_5、T_6 与 A、B 端相连,使 A="1",B="0",这样就迫使 T_2 导通,T_1 截止。当输入信号和地址选择信号消失后,T_5、T_6、T_7、T_8 截止,T_1、T_2 就保持被写入的状态不变,使得只要不掉电,写入的信息"1"就能保持不变。写入"0"的原理与此类似。

读操作时,若某个存储元被选中(X、Y 地址选择线均为高电平),则 T_5、T_6、T_7、T_8 都导通,于是存储元的信息被送到 I/O 端和$\overline{I/O}$端上。I/O 端和$\overline{I/O}$端连接到一个差动读出放大器上,从其电流方向即可判断出所存信息是"1"还是"0"。

SRAM 的使用十分方便,在微型计算机领域有着极其广泛的应用。下面就以典型的 SRAM 芯片 6264 为例,说明它的外部特性及工作过程。

1. 6264 存储芯片的引线及其功能

6264 芯片是一个 8K×8b 的 CMOS SRAM 芯片,其引脚如图 5-4 所示。6264 芯片共有 28 条引出线,包括 13 根地址信号线、8 根数据信号线以及 4 根控制信号线,它们的含义分别如下。

(1) A_0~A_{12}:13 位地址信号线。一个存储芯片上地址线的多少决定了该芯片有多

少个存储单元。13 根地址信号线上的地址信号编码最多有 2^{13} 种组合，可产生 8192(8K)个地址编码，从而保证了芯片上的 8K 个单元每单元都有唯一的地址，即芯片的 13 根地址线上的信号经过芯片的内部译码，可以决定选中 6264 芯片上 8K 个存储单元中的哪一个。在与系统连接时，这 13 根地址线通常接到系统地址总线的低 13 位上，以便 CPU 能够寻址芯片上的各个单元。

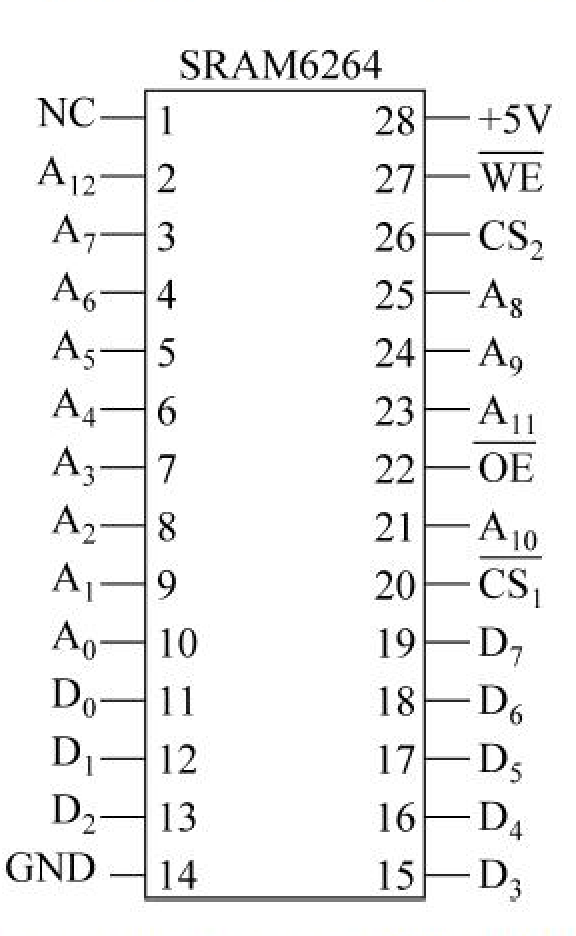


图 5-4　SRAM 6264 外部引线图

(2) $D_0 \sim D_7$：8 根双向数据线。对 SRAM 芯片来讲，数据线的根数决定了芯片上每个存储单元的二进制位数，8 根数据线说明 6264 芯片的每个存储单元中可存储 8 位二进制数，即每个存储单元有 8 位。使用时，这 8 根数据线与系统的数据总线相连。当 CPU 存取芯片上的某个存储单元时，读出和写入的数据都通过这 8 根数据线传送。

(3) $\overline{CS_1}$、CS_2：片选信号线。当$\overline{CS_1}$ 为低电平、CS_2 为高电平($\overline{CS_1}=0$，$CS_2=1$)时，该芯片被选中，CPU 才可以对其进行读写操作。不同类型的芯片，其片选信号的数量不一定相同，但要选中该芯片，必须所有的片选信号同时有效才行。事实上，一个微机系统的内存空间是由若干块存储器芯片组成的，某块芯片映射到内存空间的哪一个位置(即处于哪一个地址范围)上，是由高位地址信号决定的。系统的高位地址信号和控制信号通过译码产生片选信号，将芯片映射到所需要的地址范围上。6264 有 13 根地址线($A_0 \sim A_{12}$)，8086/8088 CPU 则有 20 根地址线，所以这里的高位地址信号就是 $A_{13} \sim A_{19}$。

(4) $\overline{OE}$：输出允许信号。只有当$\overline{OE}$为低电平时，CPU 才能够从芯片中读出数据。

(5) $\overline{WE}$：写允许信号。当$\overline{WE}$为低电平时，允许数据写入芯片；而当$\overline{WE}=1$，$\overline{OE}=0$时，允许数据从该芯片读出。

(6) 其他引线：V_{CC}为+5V 电源，GND 是接地端，NC 表示空端。

表 5-1 为芯片 4 个主要控制信号的功能表。

表 5-1　6264 真值表

$\overline{WE}$	$\overline{CS_1}$	CS_2	$\overline{OE}$	$D_0 \sim D_7$
0	0	1	×	写入
1	0	1	0	读出
× × ×	0 1 1	0 1 0	× × ×	三态 (高阻)

2. 6264 存储芯片的工作过程

对 6264 芯片的存取操作包括数据的写入和读出。

写入数据的过程是：首先把要写入单元的地址送到芯片的地址线 $A_0 \sim A_{12}$ 上；要写入的数据送到数据线上；然后使 $\overline{CS_1}$、CS_2 同时有效（$\overline{CS_1}=0$，$CS_2=1$）；再在 $\overline{WE}$ 端加上有效的低电平，$\overline{OE}$ 端状态可以任意。这样，数据就可以写入指定的存储单元中。写入过程的工作时序如图 5-5 所示。

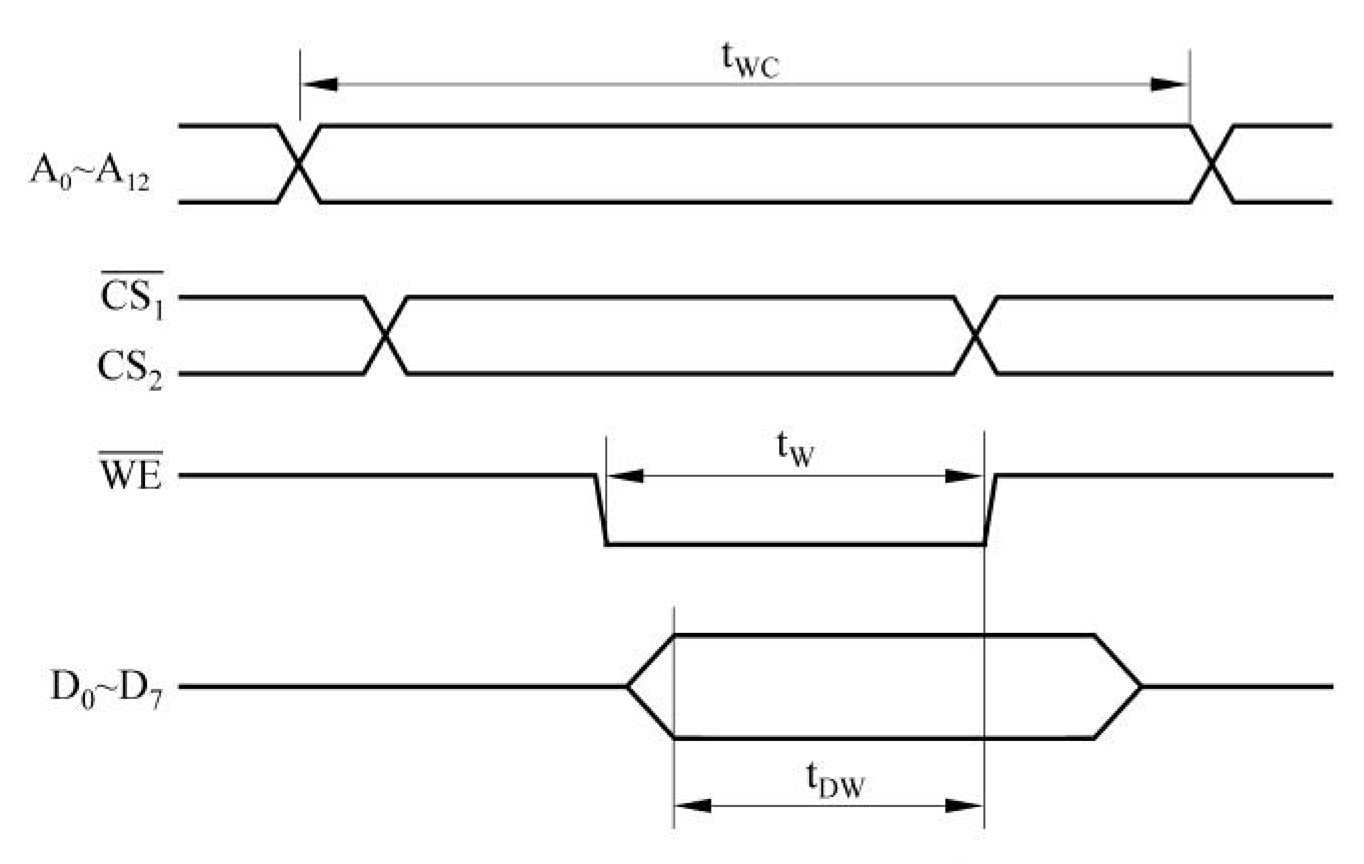


图 5-5　SRAM 6264 写操作时序图

从芯片中读出数据的过程与写操作类似：先把要读出单元的地址送到 6264 的地址线上，然后使 $\overline{CS_1}=0$ 和 $CS_2=1$ 同时有效；与写操作不同的是，此时要使读允许信号 $\overline{OE}=0$，$\overline{WE}=1$，这样，选中单元的内容就可从 6264 的数据线读出。读出过程的时序如图 5-6 所示。

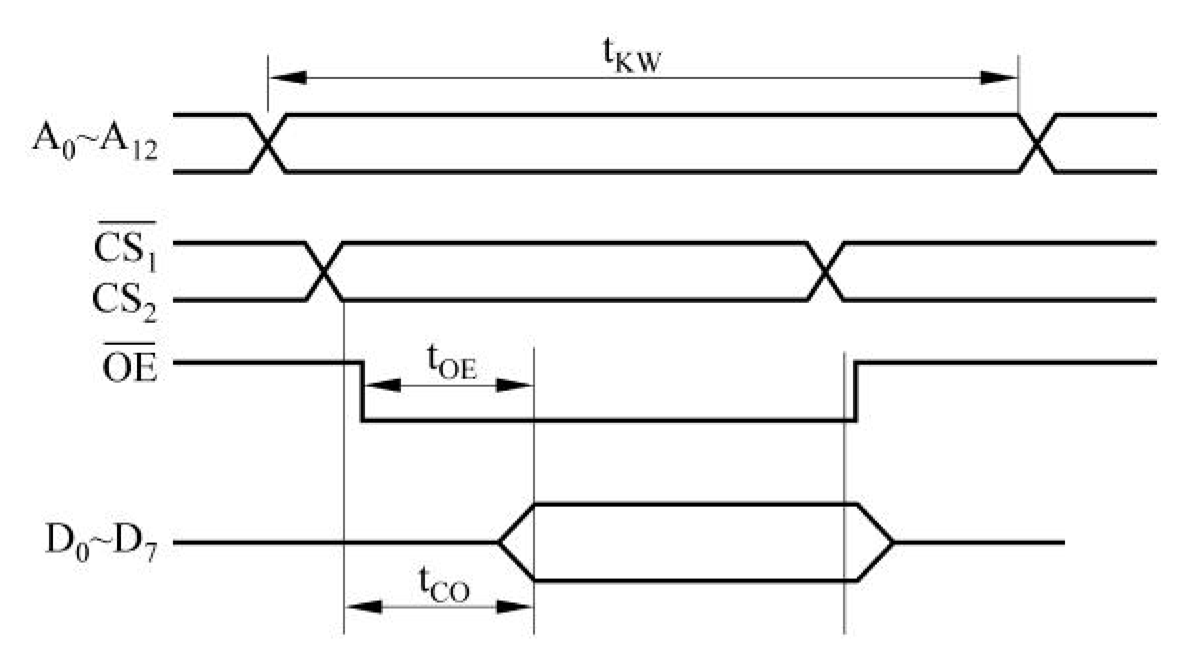


图 5-6　SRAM 6264 读操作时序图

CPU 的取指令周期和对存储器读写都有固定的时序，因此对存储器的存取速度有一定的要求。当对存储器进行读操作时，CPU 发出地址信号和读命令后，存储器必须在读允许信号有效期内将选中单元的内容送到数据总线上。同样，在进行写操作时，存储器也须在写脉冲有效期间将数据写入指定的存储单元。否则，就会出现读写错误。

如果可选择的存储器的存取速度太慢，不能满足上述要求，就需要设计者采取适当的措施来解决这一问题。最简单的解决办法就是降低 CPU 的时钟频率，即延长时钟周期 T_{CLK}，但这样做会降低系统的运行速度。另一种方法是利用 CPU 上的 READY 信号，使 CPU 在对慢速存储器操作时插入一个或几个等待周期 T_W，以等待存储器操作的完成。

当然，随着技术的发展，现有存储器芯片的存取时间已达到几个纳秒(ns)，并通过存储器系统管理技术使现代微型机系统中对内存储器的访问速度已基本能够满足使用要求。但在自行开发的系统中，对此应给予足够的重视。

6264 芯片的功耗很小(工作时为 15mW，未选中时仅 10μW)，因此在简单的应用系统中，CPU 可直接和存储器相连，不用增加总线驱动电路。

3. SRAM 芯片的应用

在对 SRAM 芯片的外部引脚功能和工作时序有一定了解之后，需要进一步掌握的是如何实现它与系统的连接。将一个存储器芯片接到总线上，除部分控制信号及数据信号线的连接外，主要是如何保证该芯片在整个内存中占据的地址范围能够满足用户的要求。前边已经讲到，芯片的片选信号是由高位地址信号和控制信号的译码产生的，事实上，正是高位地址信号决定了芯片在整个内存中占据的地址范围。

1) 地址译码

先用一个形象的例子来说明地址译码的概念。假设把存储器看成一个居住小区，那么构成存储器的存储芯片就是小区内一座一座的居民楼(假定楼号为 01～30)，而存储单元就是楼内的各个居住单元(假定单元号为 101～825)。如果某户居民住在 10 号楼 510 单元，则该住户的地址可以记为 10-510，这里的 10 就是高位地址，相当于楼号；510 就是低位地址，相当于楼内的单元号。要访问小区的 10-510 住户时，首先要找到楼号 10，这就是片选译码(选择一个存储芯片)；然后再找 510 单元，这就是片内寻址(选择一个存储单元)。片内寻址由存储芯片内部完成，使用者无须考虑。使用者要考虑的只是如何根据地址找到具体的住宅楼(芯片)。

因此，所谓译码，简单地讲就是将一组输入信号转换为一个确定的输出。在存储器技术中，译码就是将高位地址信号通过一组电路(译码器)转换为一个确定的输出信号(通常为低电平)并将其连接到存储器芯片的片选端，使该芯片被选中，从而使系统能够对该芯片上的单元进行读写操作。

8088/8086 CPU 能够寻址的内存空间为 1MB，共有 20 根地址信号线，其中高位(A_{19}～A_i)用于确定芯片的地址范围(即作为译码器的输入)，低位(A_{i-1}～A_0)用于片内寻址。由于在微机系统中，CPU 通常都工作在最大模式下，其控制信号需通过总线控制器与系统控制总线连接。因此，对存储器进行读写操作时，不是要求最小模式下的读写控制信号$\overline{RD}$和$\overline{WR}$有效，而是要求总线控制信号$\overline{MEMR}$或$\overline{MEMW}$有效。

地址译码的方式多种多样，综合起来主要可分为两种：用基本逻辑门电路构成译码器或用专门译码器进行译码。

2) 地址译码方式

存储器的地址译码方式可以分为两种：全地址译码和部分地址译码。

(1) 全地址译码方式。所谓全地址译码就是构成存储器时要使用全部 20 位地址总线信号，即所有的高位地址信号都用来作为译码器的输入，低位地址信号接存储芯片的地址输入线，从而使得存储器芯片上的每一个单元在整个内存空间中具有唯一的地址。

对 6264 芯片来讲，就是用低 13 位地址信号(A_0～A_{12})决定每个单元的片内地址，即

片内寻址；而用高 7 位地址信号(A_{13}～A_{19})决定芯片在内存中的地址边界，即作片选地址译码。

图 5-7 所示的是一片 SRAM 6264 与 8086/8088 系统的连接图。图中用地址总线的高 7 位信号(A_{13}～A_{19})作为地址译码器的输入，地址总线的低 13 位信号 A_0～A_{12} 接到芯片的 A_0～A_{12} 端，故这是一个全地址译码方式的连接。可以看出，当 A_{19}～A_{13} 为 0011111 时，译码器输出低电平，使 SRAM 6264 芯片的片选端 $\overline{CS_1}$ 有效(即表示选中该芯片)。所以，该 6264 芯片的地址范围为 3E000H～3FFFFH(低 13 位可以是从全为 0 到全为 1 之间的任何一个值)。

译码电路的构成不是唯一的，可以利用基本逻辑门电路(如“与”、“或”、“非”门等)构成，也可以利用第 1 章中介绍的 3-8 译码器 74LS138 构成。图 5-8 就是用 138 译码器实现同样地址范围的译码电路。

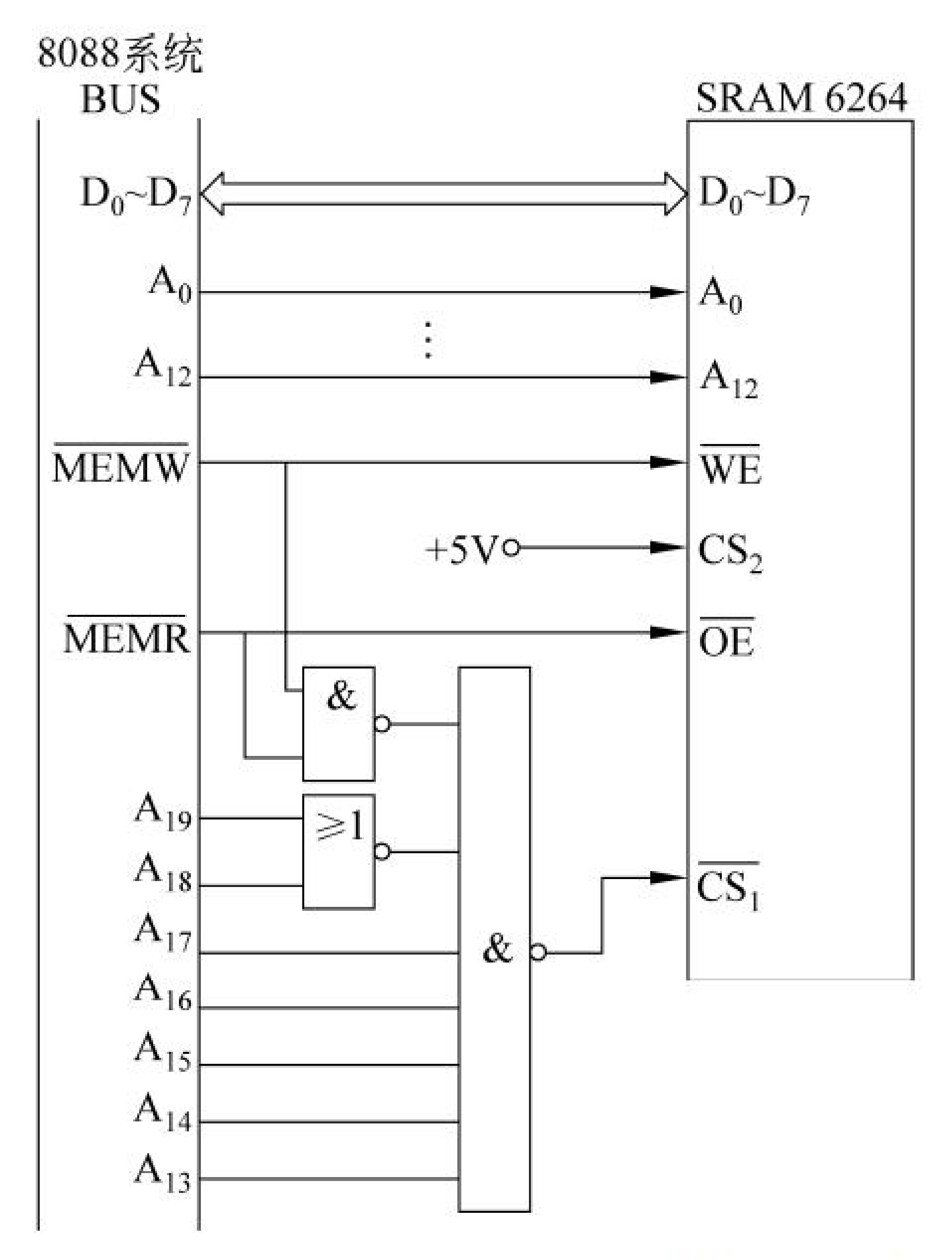


图 5-7　SRAM 6264 的全地址译码连接图

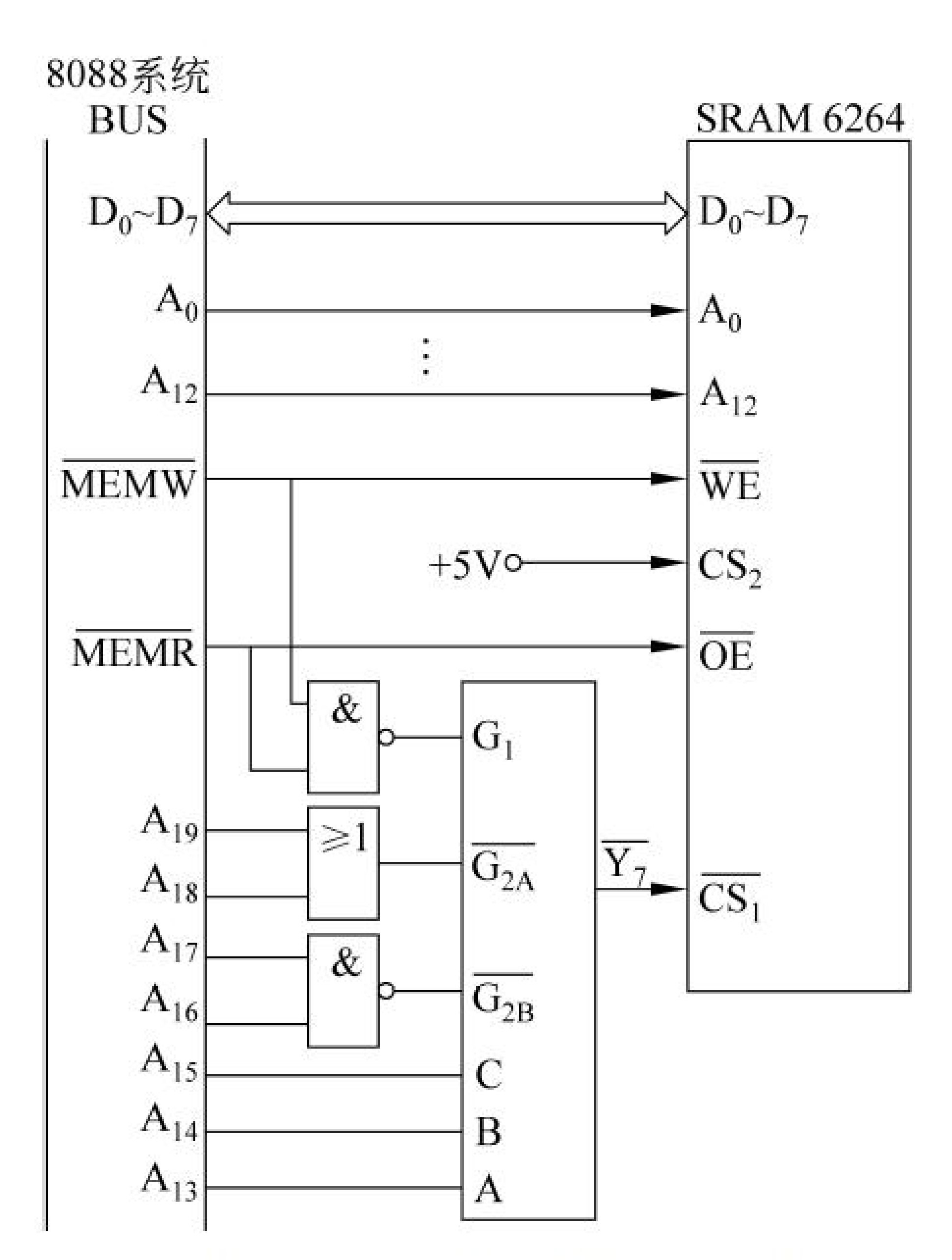


图 5-8　利用 138 译码器实现全地址译码连接

若对图 5-7 中的基本逻辑门电路进行一定的修改，如图 5-9 所示，则 6264 的地址范围就变成 C0000H～C1FFFH。由此可以看出，使用不同的译码电路可将存储器芯片映射到内存空间任意一个范围中。

(2) 部分地址译码方式。顾名思义，部分地址译码就是仅把地址总线的一部分地址信号线与存储器连接，通常是用高位地址信号的一部分(而不是全部)作为片选译码信号。图 5-10 就是一个部分地址译码的例子。从图中可以看出，该 6264 芯片被映射到了以下 4 个内存空间中：AE000H ～ AFFFFH、BE000H ～ BFFFFH、EE000H ～ EFFFFH、FE000H～FFFFFH。

该 6264 芯片共占据了 4 个 8KB 的内存地址空间，而 6264 芯片本身只有 8KB 的存储容量。为什么会出现这种情况呢？其原因就在于图中的高位地址译码并没有利用地址总

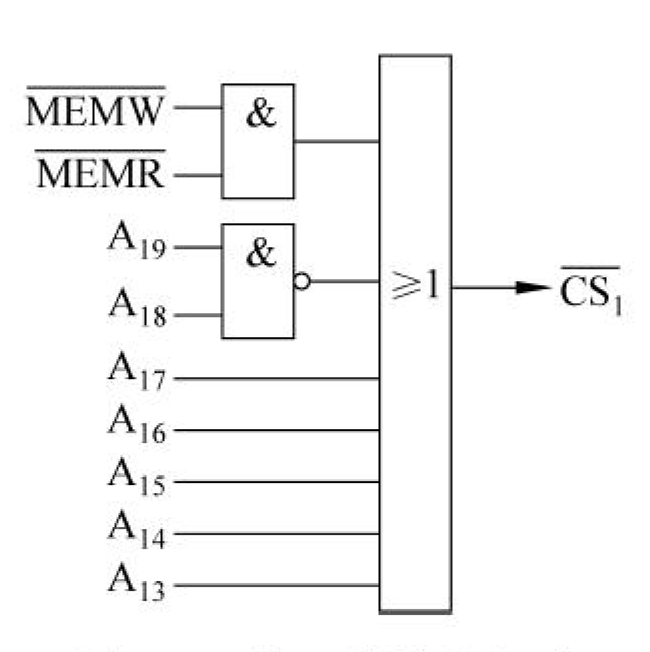


图 5-9　另一种译码电路

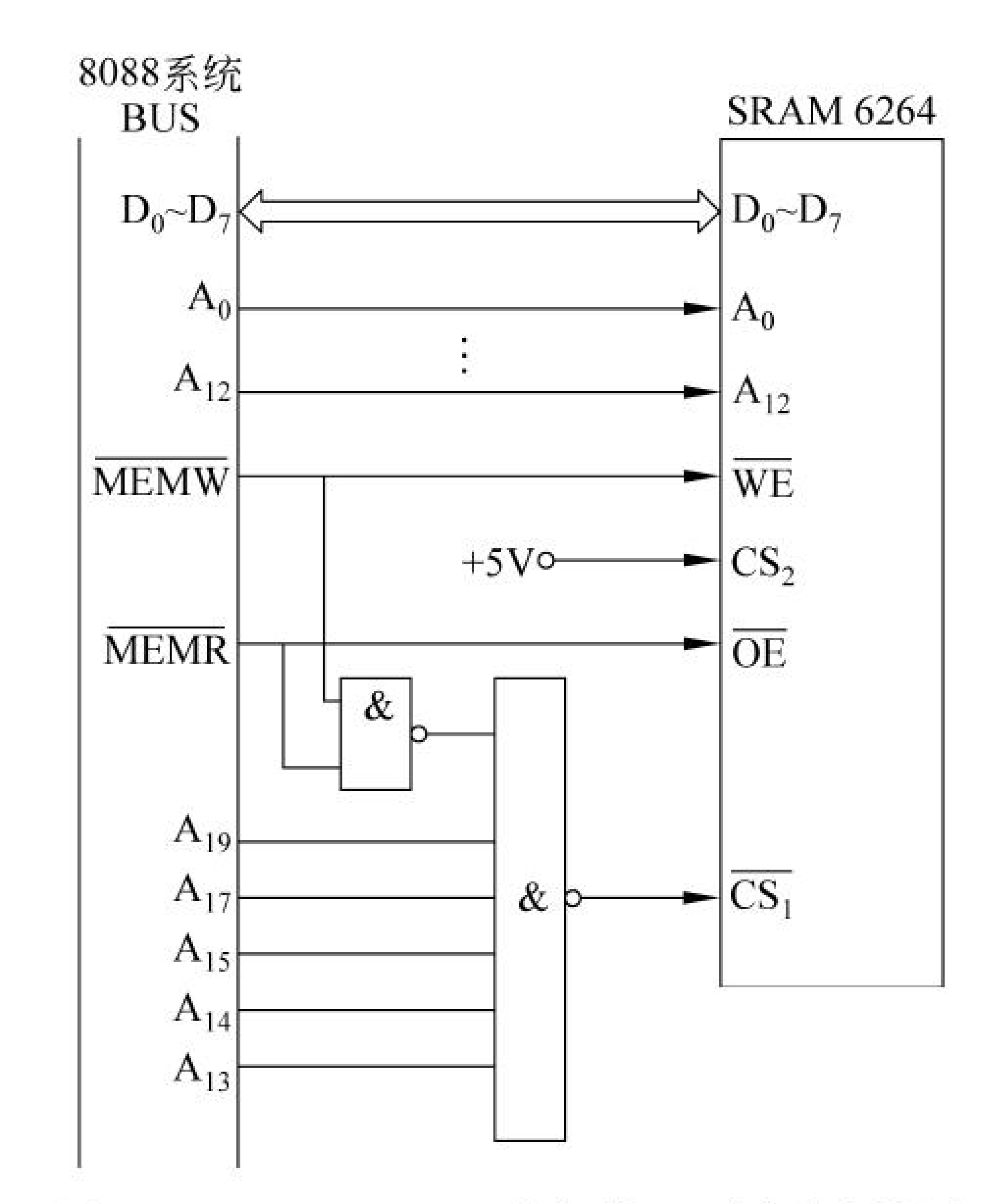


图 5-10　SRAM 6264 的部分地址译码连接图

线上的全部地址信号，而只利用了其中的一部分。在图 5-10 中，A_{18} 和 A_{16} 并未参加译码，即 A_{18} 和 A_{16} 无论是什么值都不影响译码器的输出。因此，当 A_{18} 和 A_{16} 分别为 00、01、10、11 这 4 种组合时，对应的 6264 存储芯片就占据了 4 个 8KB 的地址空间。这种只用部分地址线参加译码从而产生地址重复区的译码方式就是部分地址译码的含义。按这种地址译码方式，芯片占用的这 4 个 8KB 的区域绝不可再分配给其他芯片，否则，会造成总线竞争而使微机无法正常工作。另外，在对这个 6264 芯片进行存取时，可以使用以上 4 个地址范围的任意一个。

部分地址译码使地址出现重叠区，而重叠的部分必须空着不准使用，这就破坏了地址空间的连续性，也在实际上减小了总的可用存储地址空间。部分地址译码方式的优点是其译码器的构成比较简单、成本较低。图 5-10 中就少用两条译码输入线，但这点是以牺牲可用内存空间为代价换来的。

可以想象，参加译码的高位地址越少，译码器就越简单，而同时所构成的存储器所占用的内存地址空间就越多。若只用一条高位地址线作片选信号，如在图 5-10 中，若只将 A_{19} 接在 $\overline{CS_1}$ 上，则这片 6264 芯片将占据 00000H～7FFFFH 共 512KB 的地址空间。这种只用一条高位地址线进行片选的连接方法称为线性选择，这种地址译码方法一般仅用于系统中只使用 1～2 个存储芯片的情况。

在实际应用中，采用全地址译码还是部分地址译码应根据具体情况来定。如果地址资源很富余，为使电路简单可考虑用部分地址译码方式；如果要充分利用地址空间，则应采用全地址译码方式。

3）静态 RAM 的应用举例

以上讲述了当利用 RAM 芯片构成内存时经常采用的两种地址译码方式，其中最常使用的是全地址译码。上面已经提到，实现全地址译码可以使用各种基本逻辑门电路，也

可以用现成的译码器芯片,如 74LS138 译码器等。译码器的种类很多,如其他 74 系列芯片、PAL、GAL 等,限于篇幅这里就不一一介绍了。下面通过一个例子来说明如何使用 SRAM 芯片构成所需的存储器。

【例 5-1】 用 SRAM 6116 芯片构成地址范围在 78000H～78FFFH 之间的一个 4KB 的存储器。

SRAM 6116 芯片是 2K×8b 的存储器芯片,其外部引线如图 5-11 所示。具有 11 根地址线($A_0 \sim A_{10}$)、8 根数据线($D_0 \sim D_7$)、读写控制信号 $R/\overline{W}$(当 $R/\overline{W}=0$ 时写入,$R/\overline{W}=1$ 时读出)、输出允许信号$\overline{OE}$及片选信号$\overline{CS}$。

题目分析:

由于 SRAM 6116 的容量为 2KB,因此,要构成一个 4KB 的存储器,需要两片 6116 芯片。由题目所给的地址范围可知,其容量正好为 4KB,即表明两片存储器芯片都具有唯一的地址范围。因此,须采用全地址译码方式。

这里选用 74LS138 作为地址译码器。图 5-12 为存储器与工作在最大模式下的 8088 系统总线的连接图。图中,用 74LS138 和一些门电路构成地址译码器,对地址线高 9 位($A_{11} \sim A_{19}$)进行译码。将$\overline{MEMR}$、$\overline{MEMW}$信号组合后接到 138 译码器的使能端,保证了仅在对存储器进行读写操作时 138 译码器才能工作。

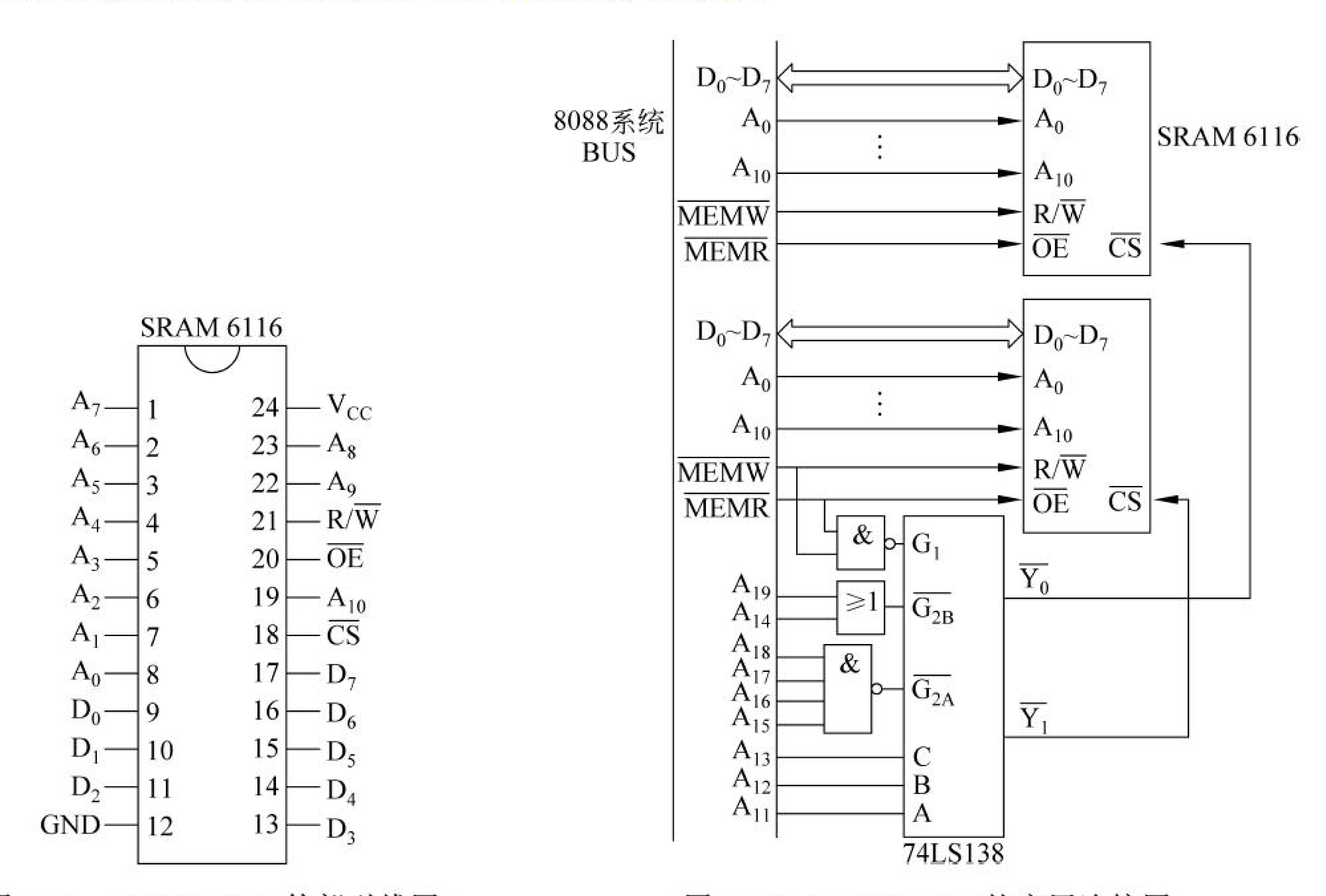


图 5-11　SRAM 6116 外部引线图

图 5-12　SRAM 6116 的应用连接图

5.2.2　动态随机存取存储器

动态随机存取存储器(DRAM)的存储元有两种结构:四管存储元和单管存储元。四

管存储元的缺点是元件多、占用芯片面积大,故集成度较低,但外围电路较简单。单管存储元的元件数量少、集成度高,但外围电路比较复杂。这里仅简单介绍一下单管存储元的存储原理。

单管动态存储元电路如图 5-13 所示,它由一个 MOS 管 T_1 和一个电容 C 构成。写入时,字选择线(地址选择线)为"1",T_1 管导通,写入的信息通过位线(数据线)存入电容 C 中;读出时,字选择线为"1",存储在 C 电容上的电荷通过 T_1 输出到位线上,根据位线上有无电流即可得知存储的信息是"1"还是"0"。

DRAM 集成度高、价格低,在微型计算机中有着极其广泛的使用。构成微机内存的内存条几乎毫无例外地都是由 DRAM 组成的。下面以一种 DRAM 芯片 2164A 为例来说明 DRAM 的外部特性及工作过程。

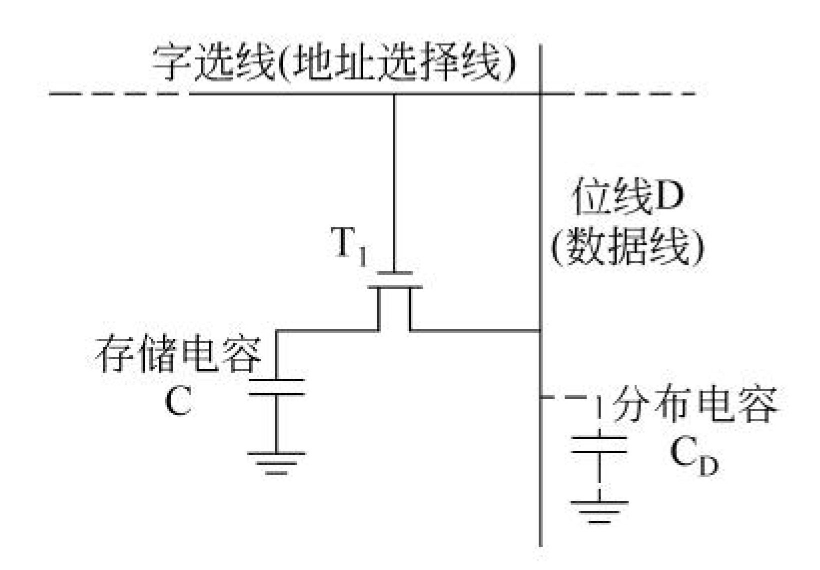


图 5-13　单管动态存储元的电路

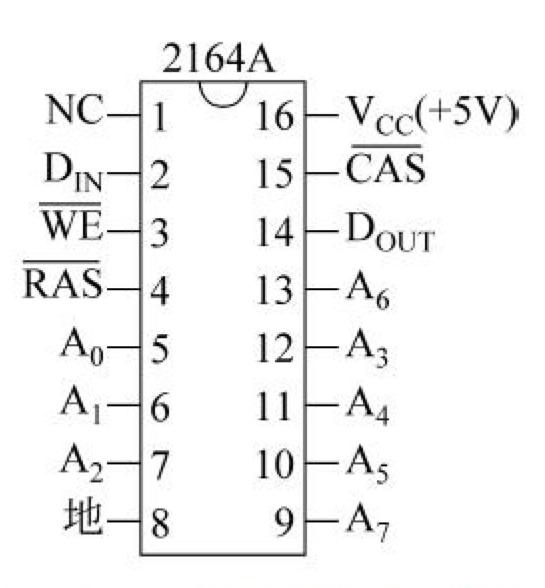


图 5-14　2164A 外部引脚图

1. 2164A 的引脚功能

2164A 是一块 64K×1b 的 DRAM 芯片,与其类似的芯片有很多种,如 3764、4164 等。图 5-14 所示为 2164A 的引脚图。

(1) A_0～A_7:地址输入线。DRAM 芯片在构造上的特点是芯片上的地址引线是复用的。虽然 2164 的容量为 64K 个单元,但它并不像对应的 SRAM 芯片那样有 16 根地址线,而是只有这个数量的一半,即 8 根地址线。那么它是如何用 8 根地址线来寻址这 64K 个单元的呢?实际上,在存取 DRAM 芯片的某单元时,其操作过程是将存取的地址分两次输入到芯片中,每一次都由同一组地址线输入。两次送到芯片上去的地址分别称为行地址和列地址。它们被锁存到芯片内部的行地址锁存器和列地址锁存器中。

可以想象,在芯片内部,各存储单元是按照矩阵结构排列的。行地址信号通过片内译码选择一行,列地址信号通过片内译码选择一列,这样就决定了选中的单元。可以简单地认为该芯片有 256 行和 256 列,共同决定 64K 个单元。对于其他 DRAM 芯片也可以按同样方式考虑。如 21256,它是 256K×1b 的 DRAM 芯片,有 256 行,每行为 1024 列。

综上所述,动态存储器芯片上的地址引线是复用的,CPU 对它寻址时的地址信号分成行地址和列地址,分别由芯片上的地址线送入芯片内部进行锁存、译码,从而选中要寻址的单元。

(2) D_{IN} 和 D_{OUT}:芯片的数据输入、输出线。其中 D_{IN} 为数据输入线,当 CPU 写芯片的某一单元时,要写入的数据由 D_{IN} 送到芯片内部;同样,D_{OUT} 是数据输出线,当 CPU 读

芯片的某一单元时，数据由此线输出。

(3) $\overline{RAS}$：行地址锁存信号。该信号将行地址锁存在芯片内部的行地址锁存器中。

(4) $\overline{CAS}$：列地址锁存信号。该信号将列地址锁存在芯片内部的列地址锁存器中。

(5) $\overline{WE}$：写允许信号。当它为低电平时，允许将数据写入；反之，当$\overline{WE}=1$时，可以从芯片读出数据。

2. DRAM 的工作过程

1）数据读出

DRAM 的数据读出过程的时序图如图 5-15 所示。首先将行地址加在 A_0～A_7 上，然后使$\overline{RAS}$行地址锁存信号有效，该信号的下降沿将行地址锁存在芯片内部；接着将列地址加到芯片的 A_0～A_7 上，再使$\overline{CAS}$列地址锁存信号有效，其下降沿将列地址锁存；然后保持$\overline{WE}=1$，则在$\overline{CAS}$有效期间（低电平），数据由 D_{OUT} 端输出并保持。

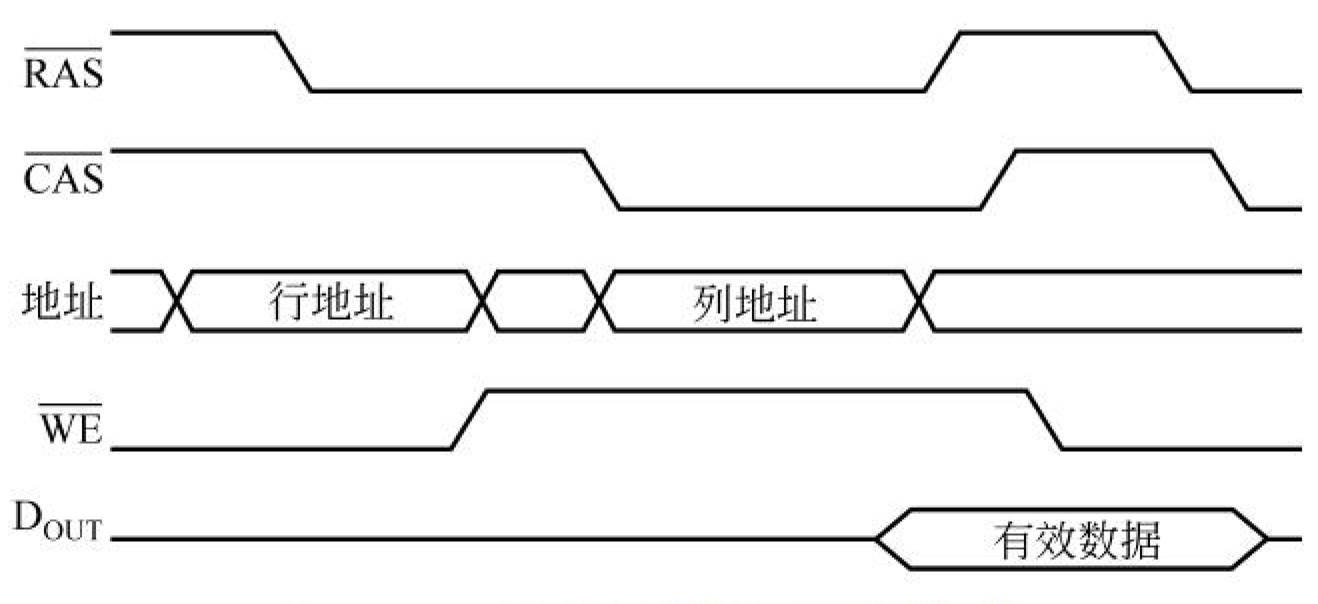


图 5-15　DRAM 的数据读出时序图

2）数据写入

DRAM 的数据写入过程的时序图如图 5-16 所示。数据写入与数据读出的过程基本类似，区别是送完列地址后，写入过程要将$\overline{WE}$端置为低电平，然后将要写入的数据从 D_{IN} 端输入。

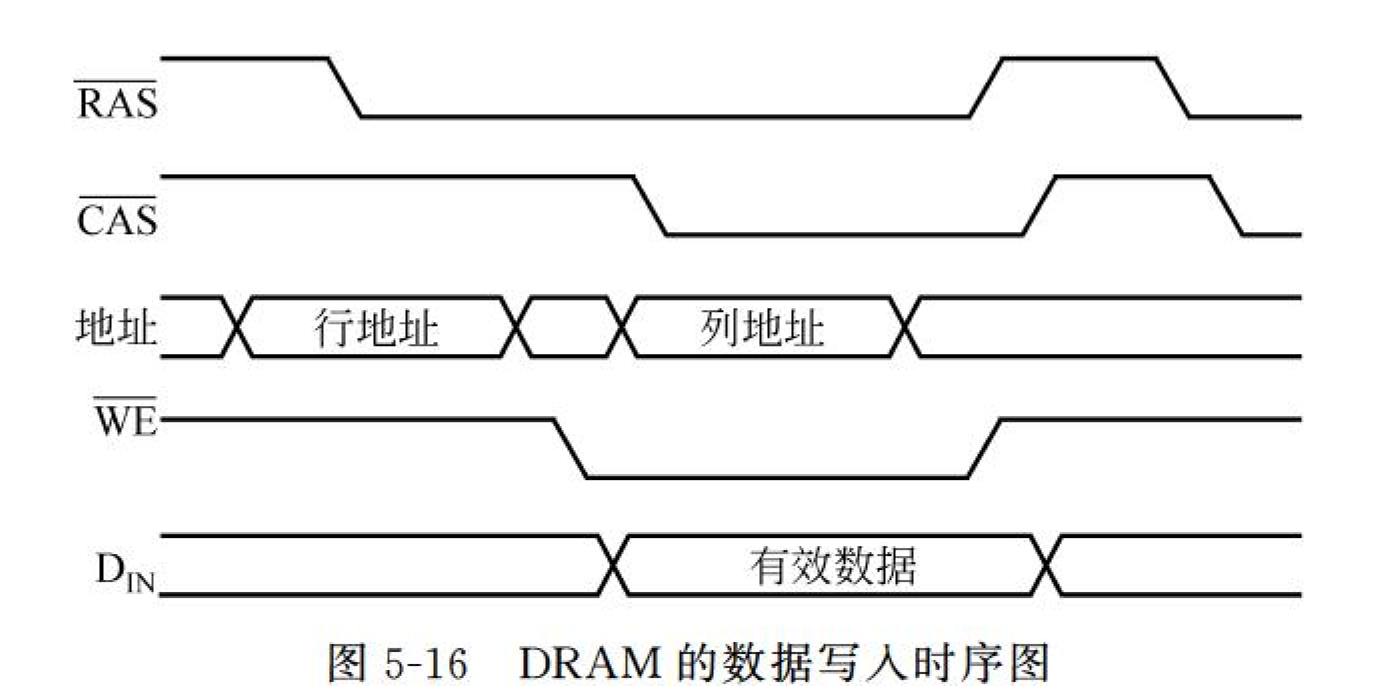


图 5-16　DRAM 的数据写入时序图

3）刷新

由于 DRAM 是靠电容来储存信息的，而电容总是存在缓慢放电现象，时间长了就会使存放的信息丢失。因此，DRAM 使用中的一个重要问题就是必须对它所存储的信息定

时进行刷新。所谓刷新就是将动态存储器中存放的每一位信息读出并重新写入的过程。刷新的方法是使列地址锁存信号无效($\overline{CAS}$=1),只送上行地址并使行地址锁存信号$\overline{RAS}$有效($\overline{RAS}$=0),然后芯片内部的刷新电路就会对所选中行上各单元中的信息进行刷新(对原来为“1”的电容补充电荷,原来为“0”的则保持不变)。每次送出不同的行地址,就可以刷新不同行的存储单元;将行地址循环一遍,就可刷新整个芯片的所有存储单元。由于刷新时$\overline{CAS}$无效,故位线上的信息不会送到数据总线上。

DRAM 芯片的刷新时序图如图 5-17 所示。图中$\overline{CAS}$保持无效,利用$\overline{RAS}$锁存刷新的行地址进行逐行刷新。DRAM 要求每隔 2～8ms 刷新一次,这个时间称为刷新周期。在刷新周期中,DRAM 是不能进行正常的读写操作的,这一点由刷新控制电路予以保证。

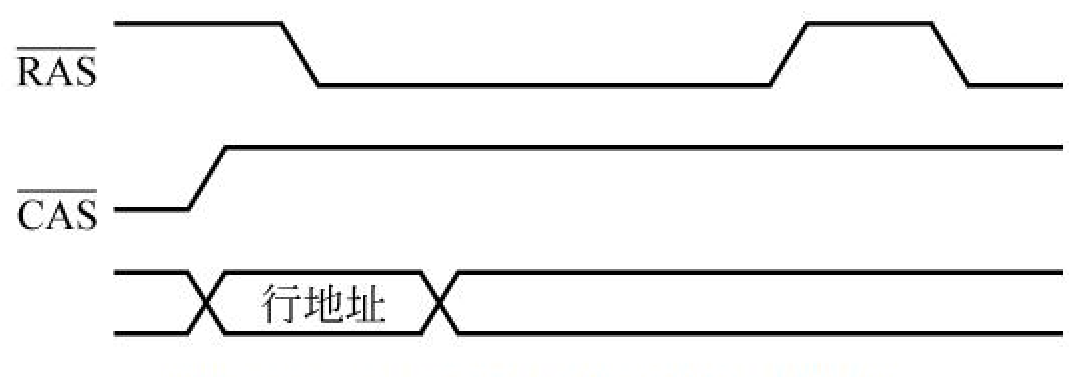


图 5-17　DRAM 芯片的刷新时序图

3. DRAM 在系统中的连接

现在微型机系统中,大多采用 DRAM 芯片构成主存储器。由于在使用中既要做到能够正确读写,又要能在规定的时间里对它进行刷新,因此,微型机中对 DRAM 的连接和控制电路要比 SRAM 复杂得多。这里仅通过一个简化的电路示意图来说明 DRAM 的使用。

图 5-18 所示的是 PC/XT 微型机的 DRAM 简化电路图。图中用虚线画的长方体表示由 8 片(加奇偶校验位则为 9 片)2164A DRAM 组成的 64KB 的存储器。LS158 是二选一的数据选择器,LS245 为驱动器。当 CPU 读写存储器的某个单元时,首先由行列锁存信号电路送出行地址锁存信号$\overline{RAS}$,同时 ADDSEL=0,使 LS158 的 A 端口导通,CPU

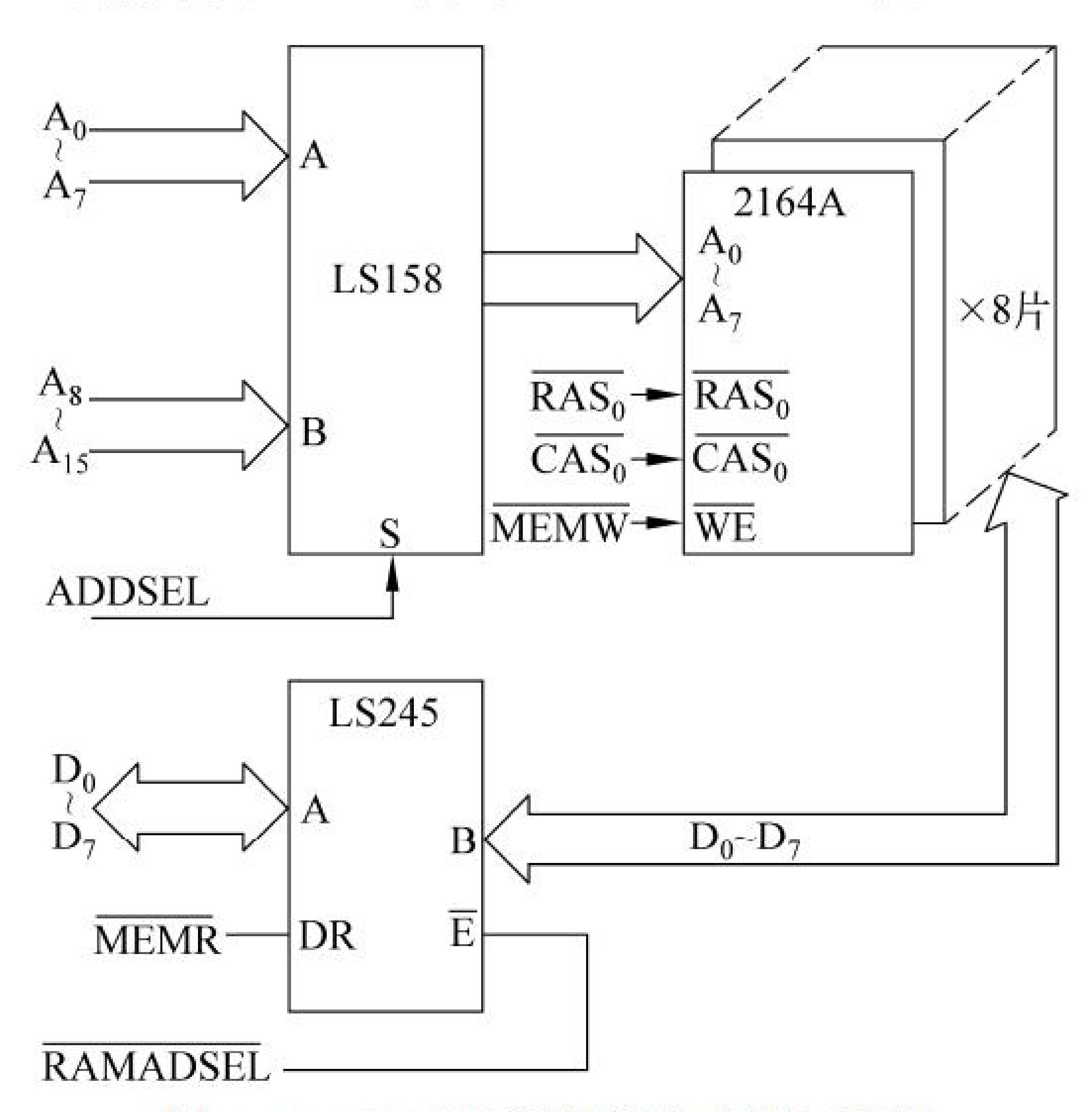


图 5-18　DRAM 读写简化电路示意图

将 8 位行地址信号通过地址总线的低 8 位($A_0 \sim A_7$)从 LS158 的 A 口加到存储器芯片上,并在$\overline{RAS}$作用下锁存于存储芯片内部的行地址锁存器。60ns 后,ADDSEL=1,使 LS158 的 B 端口导通,CPU 将 8 位列地址信号通过地址总线的 $A_8 \sim A_{15}$ 从 LS158 的 B 口加到存储器芯片上,延迟 40ns 后由$\overline{CAS}$将其锁存于存储芯片内部的列地址锁存器。最后,在存储器读写信号$\overline{MEMR}/\overline{MEMW}$控制下实现数据的读写。

PC/XT 微型机中 DRAM 的刷新过程是利用 DMA 来实现的。首先由可编程定时器 8253 每隔 15.12μs 产生一次 DMA 请求;之后由 DMA 控制器 8237 在其$\overline{DAK_0}$端产生一个低电平,使列地址锁存信号$\overline{CAS}$为高电平,而行地址信号$\overline{RAS}$为低电平;最后,通过 DMA 控制器送出刷新的行地址,实现一次刷新。

5.2.3 存储器扩展技术

任何存储芯片的存储容量都是有限的。要构成一定容量的内存,往往单个芯片不能满足字长或存储单元个数的要求,甚至字长、存储单元数都不能满足要求。这时就需要用多个存储芯片进行组合,以满足对存储容量的需求。这种组合就称为存储器的扩展,扩展时要解决的问题包括位扩展、字扩展和字位扩展。

1. 位扩展

一块实际的存储芯片,其每个单元的位数(即字长)往往与实际内存单元字长并不相等。存储芯片可以是 1 位、4 位或 8 位的,如 DRAM 芯片 Intel 2164A 为 64K×1b,SRAM 芯片 Intel 2114A 为 1K×4b,Intel 6264 芯片则为 8K×8b。而计算机中内存一般是按字节来进行组织的,若要使用 2164A、2114A 这样的存储芯片来构成内存,单个存储芯片字长(位数)就不能满足要求,这时就需要进行位扩展,以满足字长的要求。

位扩展构成的存储器系统的每个单元中的内容被存储在不同的存储器芯片上。例如,用 2 片 4K×4b 的存储器芯片经位扩展构成 4KB 的存储器中,每个单元中的 8 位二进制数被分别存在两个芯片上,即一个芯片存该单元内容的高 4 位,另一个芯片存该单元内容的低 4 位。

可以看出,位扩展保持总的地址单元数(存储单元个数)不变,但每个单元中的位数增加。

位扩展的电路连接方法是:将每个存储芯片的地址线和控制线(包括片选信号线、读写信号线等)全部并联在一起,而将它们的数据线分别引出至数据总线的不同位上,如图 5-19 所示。

【例 5-2】 用 Intel 2164A 芯片构成容量为 64KB 的存储器。

解:因为 2164A 是 64K×1b 的芯片,其存储单元数已满足要求,只是字长不够,所以需要 8 片 2164A 进行位扩展。线路连接如图 5-20 所示。图中,8 个 2164A 的数据线分别连接到数据总线的 $D_0 \sim D_7$。地址线和控制线等均按照信号名称全部并联在一起。

2. 字扩展

字扩展是对存储器容量的扩展(或存储空间的扩展)。此时存储芯片上每个存储单元

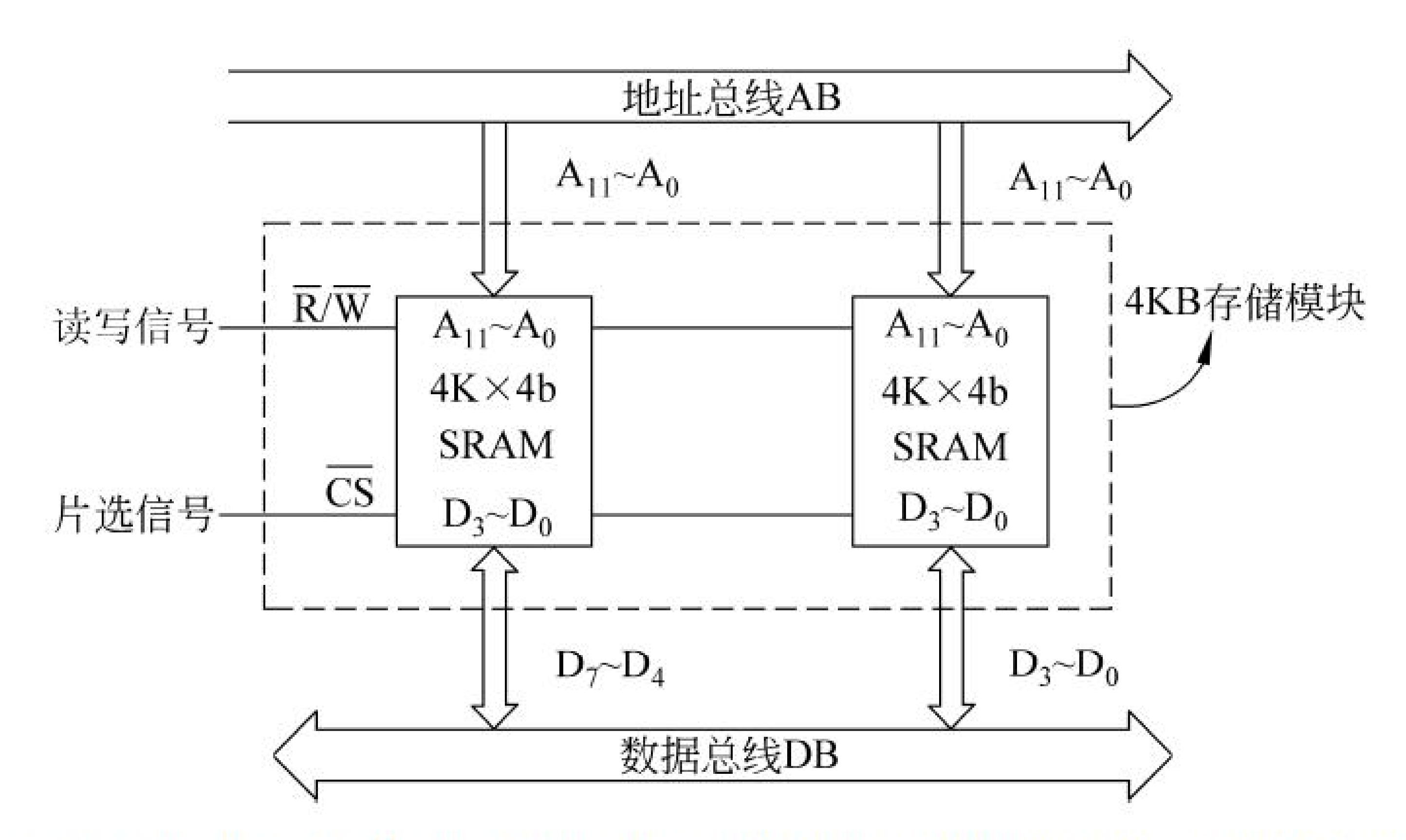


图 5-19　用 4K×4b 的 SRAM 芯片进行位扩展以构成容量为 4KB 的存储器

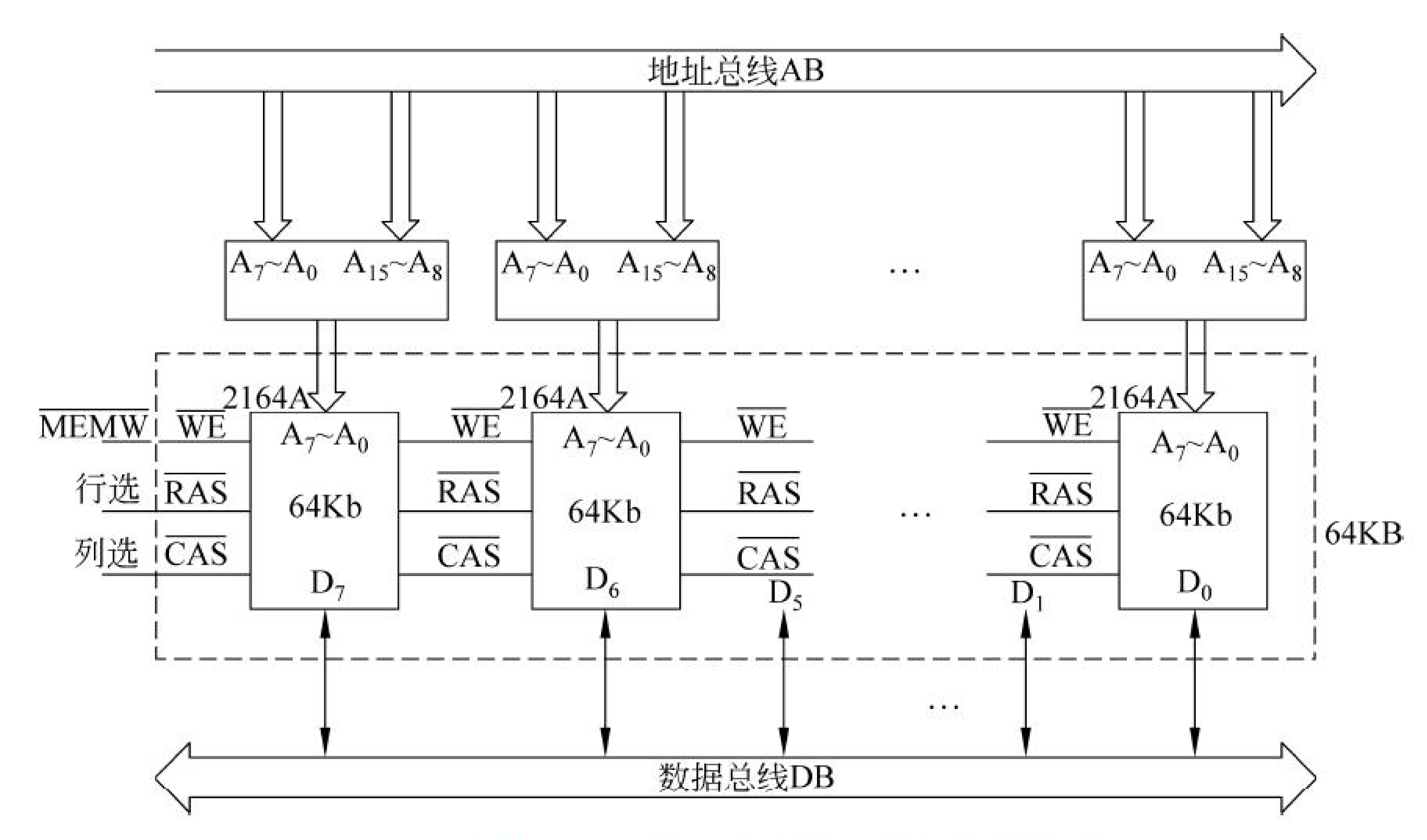


图 5-20　用 2164 A 构成容量为 64KB 的存储器

的字长已满足要求(如字长已为 8 位),只是存储单元的个数不够,需要增加的是存储单元的数量。这就是字扩展,即用多片字长为 8 位的存储芯片构成所需要的存储空间。

例如,用 2K×8b 的存储器芯片组成 4K×8b 的内存储器。在这里,字长已满足要求,只是容量不够,所以需要进行的是字扩展,显然,对现有的 2K×8b 芯片存储器,需要用两片来实现。

字扩展的电路连接方法是:将每个芯片的地址信号、数据信号和读写信号等控制信号线按信号名称全部并联在一起,只将片选端分别引出到地址译码器的不同输出端,即用片选信号来区别各个芯片的地址。其连接示意图如图 5-21 所示。

【例 5-3】 用两片 64K×8b 的 SRAM 芯片构成容量为 128KB 的存储器。

解:这里现有的芯片容量为 64KB,构成容量为 128KB 的存储器需要 128KB/64KB=2 片。线路连接如图 5-22 所示。图中两片芯片的地址范围分别为 20000H～2FFFFH 和 30000H～3FFFFH。

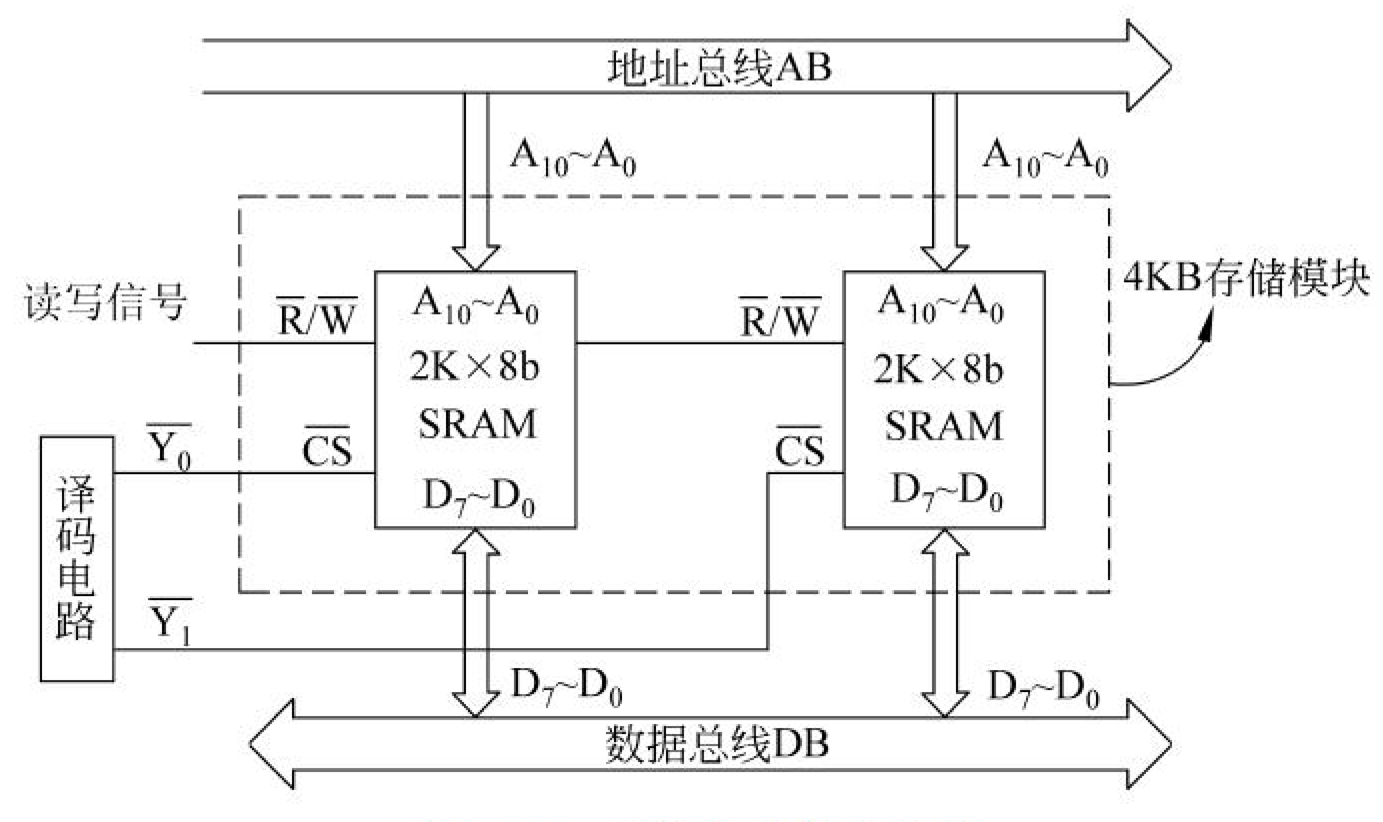


图 5-21　字扩展连接示意图

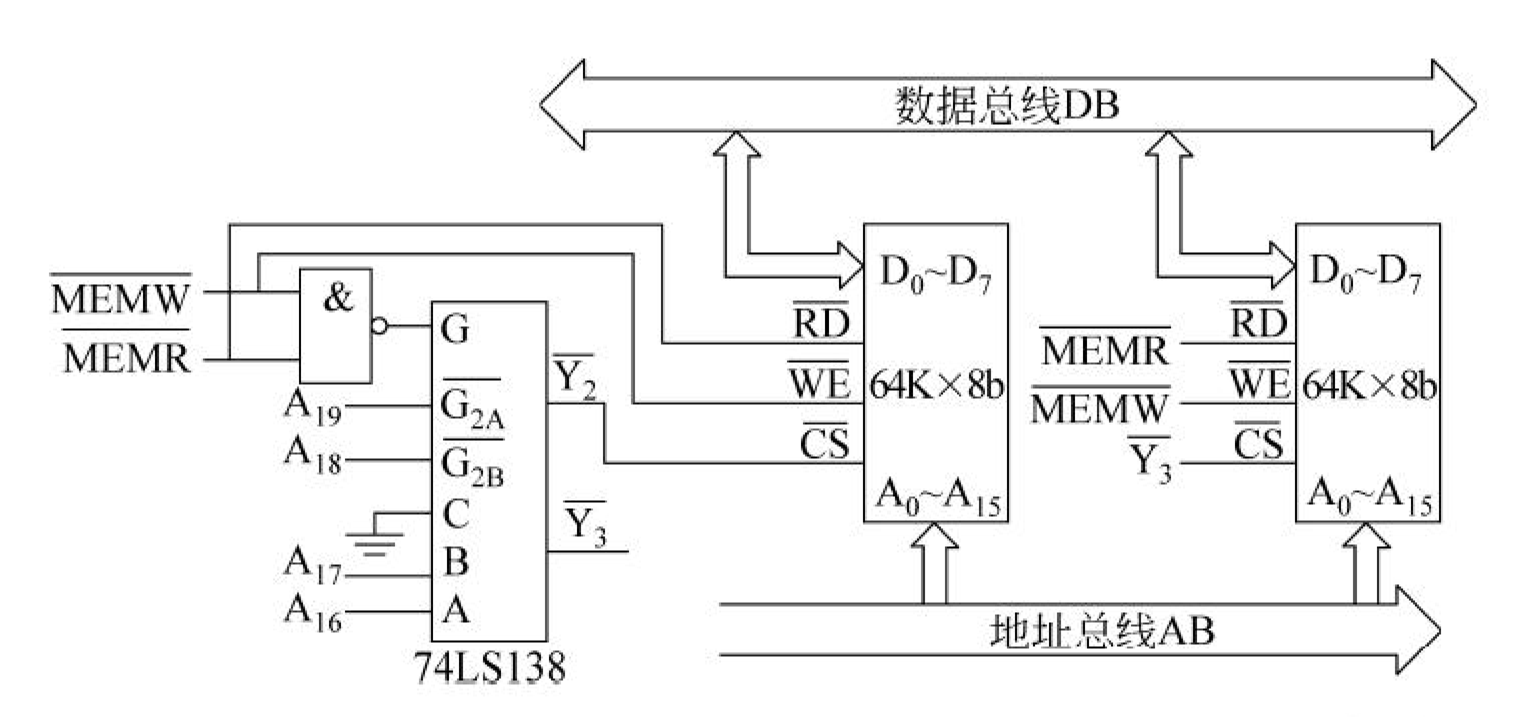


图 5-22　用 2 片 64K×8b 的 SRAM 芯片构成容量为 128KB 的存储器

3. 字位扩展

在构成一个实际的存储器时，往往需要同时进行位扩展和字扩展才能满足存储容量的需求。扩展时需要的芯片数量可以这样计算：要构成一个容量为 $M \times N$ 位的存储器，若使用 $l \times k$ 位的芯片（($l < M, k < N$)），则构成这个存储器需要 $(M/l) \times (N/k)$ 个这样的存储器芯片。

微型机中内存的构成就是字位扩展的一个很好的例子。首先，存储器芯片生产厂制造出一个个单独的存储芯片，如 64M×1b、128M×1b 等；然后，内存条生产厂将若干个芯片用位扩展的方法组装成内存模块（即内存条），如用 8 片 128M×1b 的芯片组成 128MB 的内存条；最后，用户根据实际需要购买若干个内存条插到主板上构成自己的内存系统，即字扩展。一般来讲，最终用户做的都是字扩展（即增加内存地址单元）的工作。

进行字位扩展时，一般先进行位扩展，构成字长满足要求的内存模块，然后再用若干个这样的模块进行字扩展，使总存储容量满足要求。

【例 5-4】 用 Intel 2164A 构成容量为 128KB 的内存。

解：由于 2164A 是 64K×1b 的芯片，所以首先要进行位扩展。用 8 片 2164A 组成 64KB 的内存模块，然后再用两组这样的模块进行字扩展。所需的芯片数为（128/64）×

(8/1)＝16 片。

要寻址 128K 个内存单元至少需要 17 位地址信号线($2^{17}=128K$)。而 2164A 有 64K 个单元,只需要 16 位地址信号(分为行和列),余下的 1 根地址线用于区分两个 64KB 的存储模块。

所以,构成此内存共需 16 片 2164A 芯片;至少需要 17 根地址信号线,其中 16 根用于 2164A 的片内寻址(行、列地址),1 根用于片选地址译码(用于区分存取哪一个 64KB 模块)。线路连接示意图如图 5-23 所示。

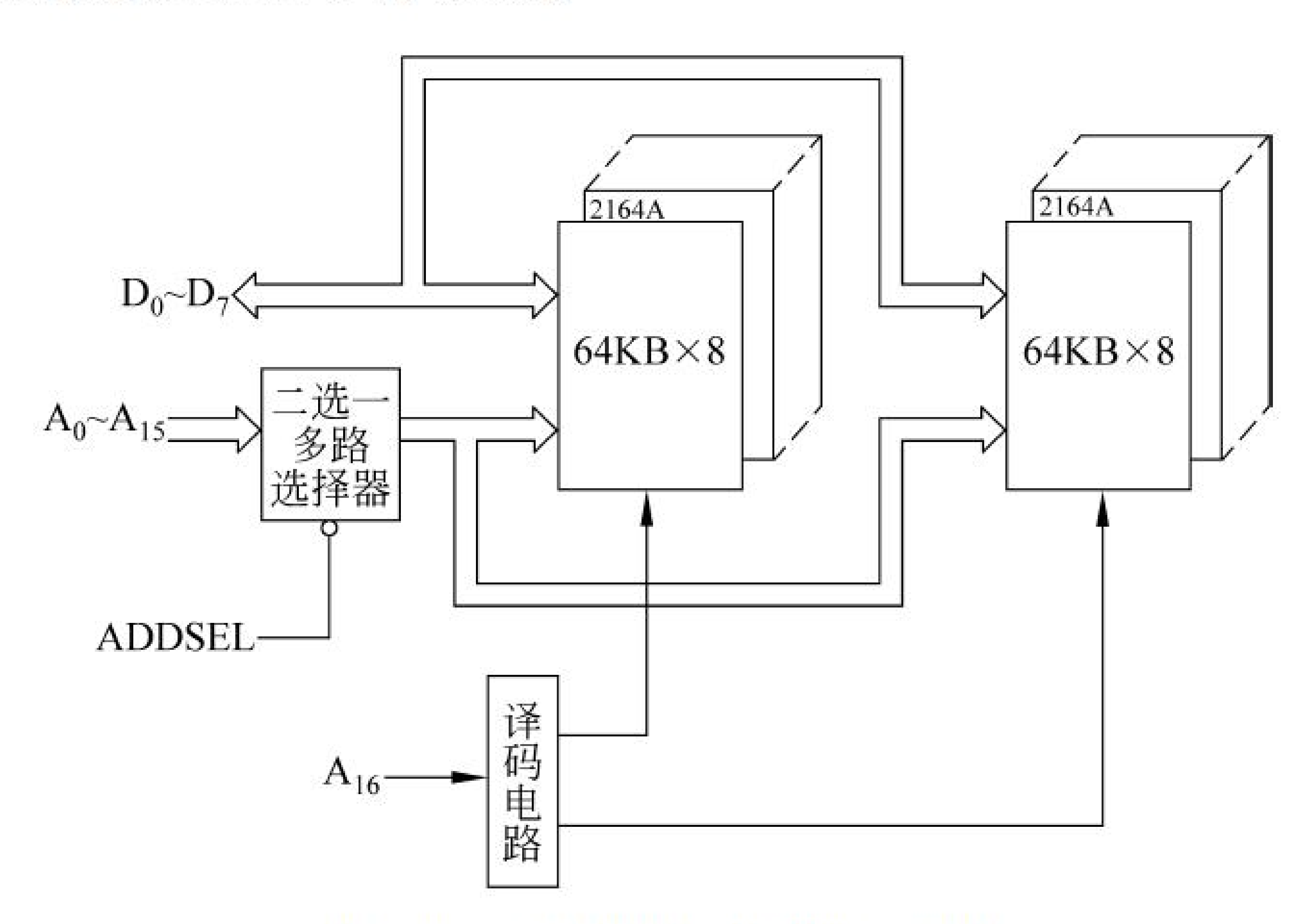


图 5-23　字位扩展应用举例示意图

综上所述,存储器容量的扩展可以分为 3 步:①选择合适的芯片;②根据要求将芯片"多片并联"进行位扩展,设计出满足字长要求的"存储模块";③对"存储模块"进行字扩展,构成符合要求的存储器。

5.3　只读存储器

只读存储器(ROM)因其具有掉电后信息不丢失的特点,故一般用于存放一些固定的程序,如监控程序、BIOS 程序等。本节主要介绍两种可擦除的只读存储器:EPROM 和 EEPROM。

5.3.1　EPROM

EPROM 是一种可擦除可编程的只读存储器。擦除时,用紫外线照射芯片上的窗口即可清除存储的内容。擦除后的芯片可以使用专门的编程写入器对其重新编程(写入新的内容)。存储在 EPROM 中的内容能够长期保存达几十年之久,而且掉电后其内容也不会丢失。下面以一种典型的 EPROM 芯片 2764 为例来介绍这类芯片的特点和应用。

1．引线及功能

2764 的外部引线如图 5-24 所示。这是一块 8K×8b 的 EPROM 芯片，它的引线与前面介绍的 SRAM 芯片 6264 是兼容的。这样的设计给使用者带来很大的方便，因为在软件调试过程中，程序经常需要修改，此时可将程序先放在 6264 中，读写修改都很方便。调试成功后，将程序固化在 2764 中，由于它与 6264 的引脚兼容，所以可以把 2764 直接插在原 6264 的插座上。这样，程序就不会由于断电而丢失。

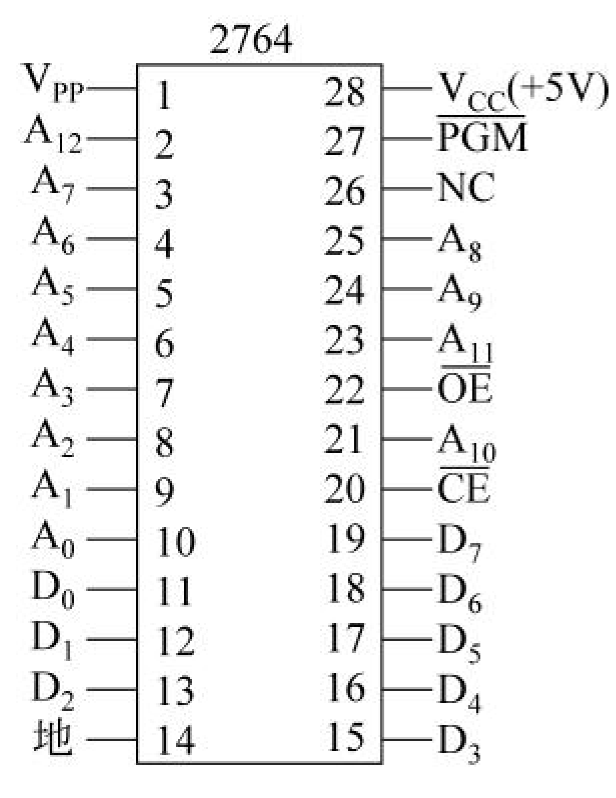


图 5-24　EPROM 2764 引线图

2764 各引脚的含义如下。

(1) $A_0 \sim A_{12}$：13 根地址输入线，用于寻址片内的 8K 个存储单元。

(2) $D_0 \sim D_7$：8 根双向数据线，正常工作时为数据输出线，编程时为数据输入线。

(3) $\overline{CE}$：片选信号，低电平有效。当$\overline{CE}$=0 时表示选中此芯片。

(4) $\overline{OE}$：输出允许信号，低电平有效。当$\overline{OE}$=0 时，芯片中的数据可由 $D_0 \sim D_7$ 端输出。

(5) $\overline{PGM}$：编程脉冲输入端。对 EPROM 编程时，在该端加上编程脉冲。读操作时$\overline{PGM}$=1。

(6) V_{PP}：编程电压输入端。编程时应在该端加上编程高电压，不同的芯片对 V_{PP} 的值要求的不一样，可以是+12.5V、+15V、+21V、+25V 等。

2．2764 的工作过程

2764 可以工作在数据读出、编程写入和擦除 3 种方式下。

1) 数据读出

数据读出是 2764 的基本工作方式，用于读出 2764 中存储的内容。其工作过程与 RAM 芯片类似，即先把要读出的存储单元地址送到 $A_0 \sim A_{12}$ 地址线上，然后使 $\overline{CE}$=0，$\overline{OE}$=0，就可在芯片的 $D_0 \sim D_7$ 上读出需要的数据。读出过程的时序图如图 5-25 所示。

因为 2764 与 6264 SRAM 在引脚上是兼容的，所以在与系统的连接使用上可按与 RAM 芯片相同的方法来进行电路设计。只是在读方式下，编程脉冲输入端$\overline{PGM}$及编程电压 V_{PP}端都接在+5V 电源 V_{CC}上。图 5-26 是 2764 芯片与 8088 总线的连接图。由图可以看出，2764 芯片的地址范围为 70000H～71FFFH。

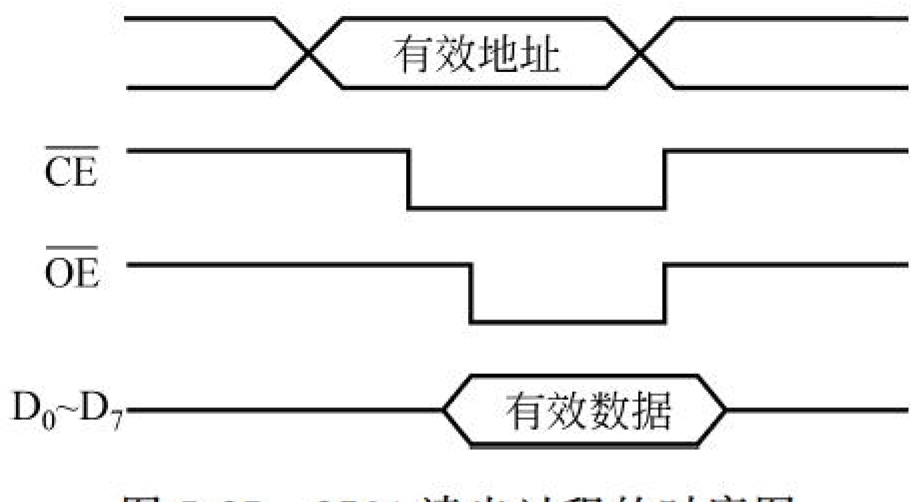


图 5-25　2764 读出过程的时序图

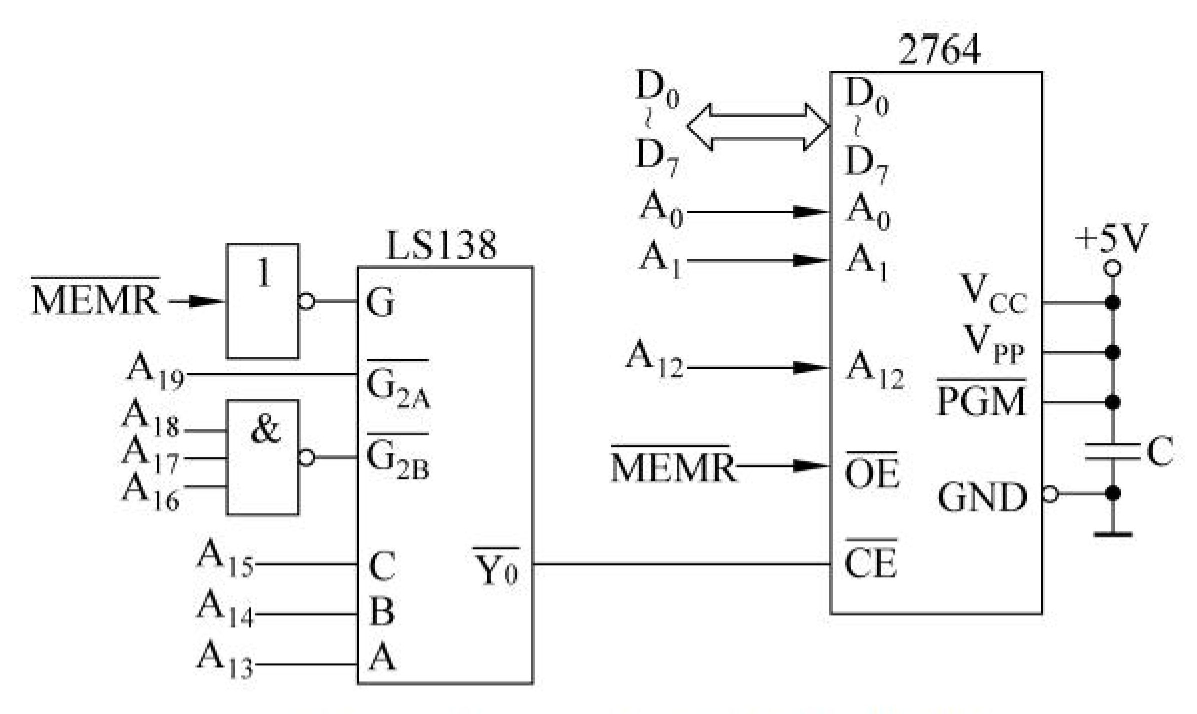


图 5-26　2764 与 8088 系统的连接图

2）编程写入

对 EPROM 芯片的编程可以有两种方式：标准编程和快速编程。

（1）标准编程方式是每给出一个编程负脉冲就写入一个字节的数据，具体的方法是：V_{CC}接+5V，V_{PP}加上芯片要求的高电压；在地址线 A_0～A_{12}上给出要编程存储单元的地址，然后使$\overline{CE}$=0，$\overline{OE}$=1；并在数据线上给出要写入的数据。上述信号稳定后，在$\overline{PGM}$端加上 50±5ms 的负脉冲，就可将一个字节的数据写入相应的地址单元中。不断重复这个过程，就可将要写的数据逐一写入对应的存储单元中。

如果其他信号状态不变，只是在每写入一个单元的数据后将$\overline{OE}$变低，则可以立即对刚写入的数据进行校验，当然也可以写完所有单元后再统一进行校验。若检查出写入数据有错，则必须全部擦除，再重新开始上述的编程写入过程。

早期的 EPROM 采用的都是标准编程方法。这种方法有两个严重的缺点：①编程脉冲太宽（约 50ms），从而使编程时间太长，对于容量较大的 EPROM，其编程的时间将长得令人难以接受，例如，对 256KB 的 EPROM，其编程时间长达 3.5 个小时以上；②不够安全，编程脉冲太宽会使芯片功耗过大而损坏 EPROM。

（2）快速编程与标准编程的工作过程一样，只是编程脉冲要窄得多。例如，EPROM 27C040 芯片的编程脉冲宽度仅为 100μs，其时序图如图 5-27 所示。其编程过程为：先用 100μs 编程脉冲依次写完所有要编程的单元，然后从头开始校验每个写入的字节。若写

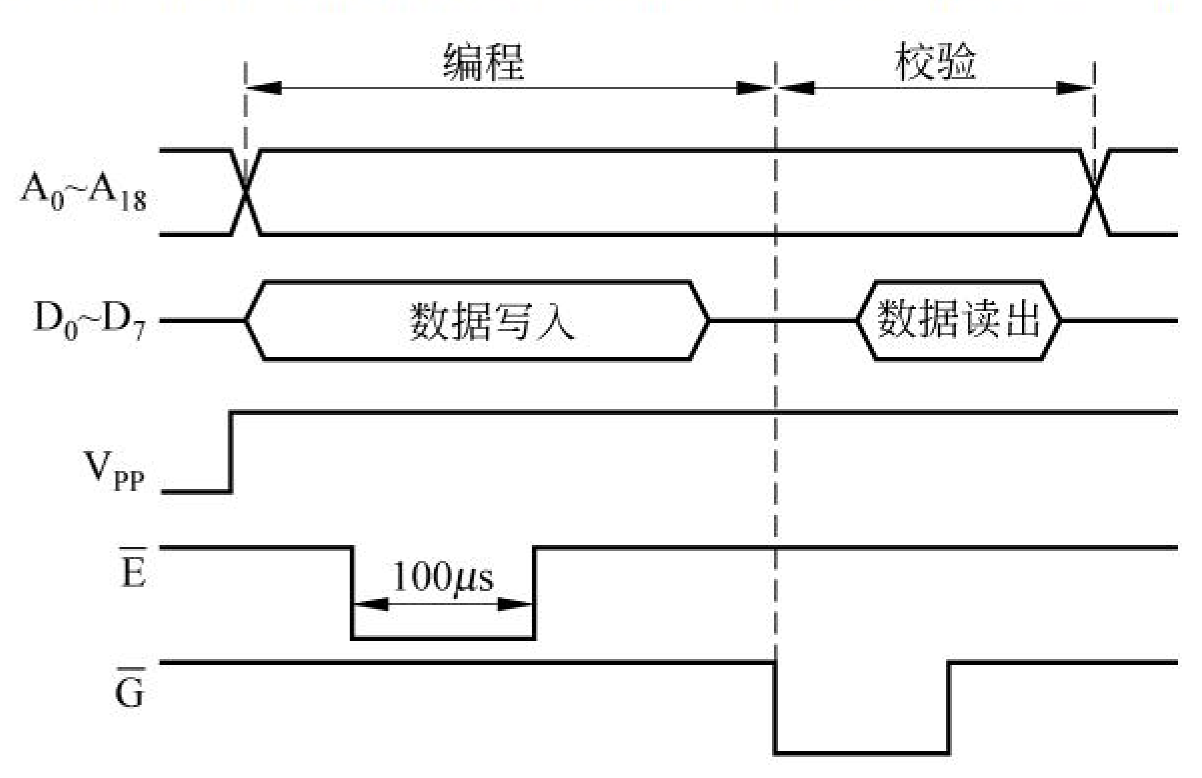


图 5-27　EPROM 27C040 的编程时序图

得不正确，则重写此单元；写完后再校验，不正确还可再写；若连续10次仍不正确，则认为芯片已损坏；最后再从头到尾对每一个编程单元校验一遍，全对，则编程即告结束。27C040的快速编程过程的流程图如图5-28所示。

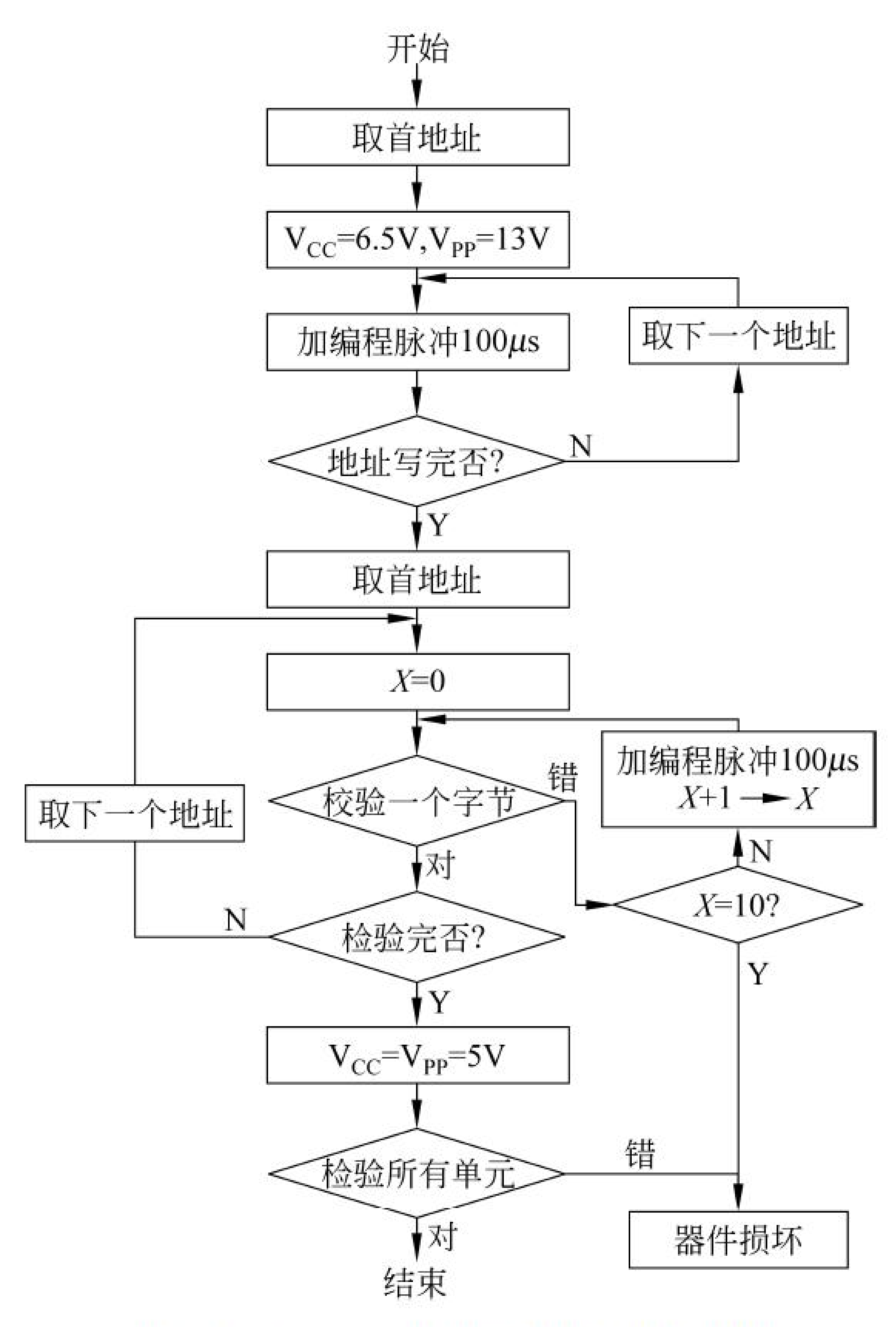


图5-28　27C040的快速编程过程流程图

有一点要注意，不同厂家、不同型号的EPROM芯片对编程的要求不一定是相同的，编程脉冲的宽度也不一样，但编程的思想是相同的。

3）擦除

EPROM的一个重要优点是可以擦除重写，而且允许擦除的次数超过上万次。一片新的或擦除干净的EPROM芯片，其每一个存储单元的内容都是FFH。要对一个使用过的EPROM进行编程，则首先应将其放到专门的擦除器上进行擦除操作。擦除器利用紫外线光照射EPROM的窗口，一般经过15～20min即可擦除干净。擦除完毕后可读一下EPROM的每个单元，若其内容均为FFH，就认为擦除干净了。

5.3.2　EEPROM

由于EEPROM(E^2PROM)(电擦除可编程只读存储器)采用电擦除技术，所以它允许在线编程写入和擦除，而不必像EPROM芯片那样需要从系统中取下来，用专门的编

程写入器编程和专门的擦除器擦除。从这一点讲，它的使用要比 EPROM 方便。另外，EPROM 虽可多次编程写入，但整个芯片只要有一位写错，也必须从电路板上取下来全部擦掉重写，这给实际使用带来很大不便。因为在实际使用中，多数情况下需要的是以字节为单位的擦除和重写，而 EEPROM 在这方面就具有很大的优越性。下面以一个典型的 EEPROM 芯片 NMC98C64A 为例介绍 EEPROM 的工作过程和应用。

1. 98C64A 的引线

NMC98C64A 为 8K×8 位的 EEPROM，其引线如图 5-29 所示。其中各引线含义如下。

（1）A_0～A_{12}为地址线，用于选择片内的 8K 个存储单元。

（2）D_0～D_7 为 8 条数据线。

（3）$\overline{CE}$为片选信号，低电平有效。当$\overline{CE}$＝0 时选中该芯片。

（4）$\overline{OE}$为输出允许信号。当$\overline{CE}$＝0，$\overline{OE}$＝0，$\overline{WE}$＝1 时，可将选中的地址单元的数据读出。这点与 6264 很相似。

（5）$\overline{WE}$是写允许信号。当$\overline{CE}$＝0，$\overline{OE}$＝1，$\overline{WE}$＝0 时，可以将数据写入指定的存储单元。

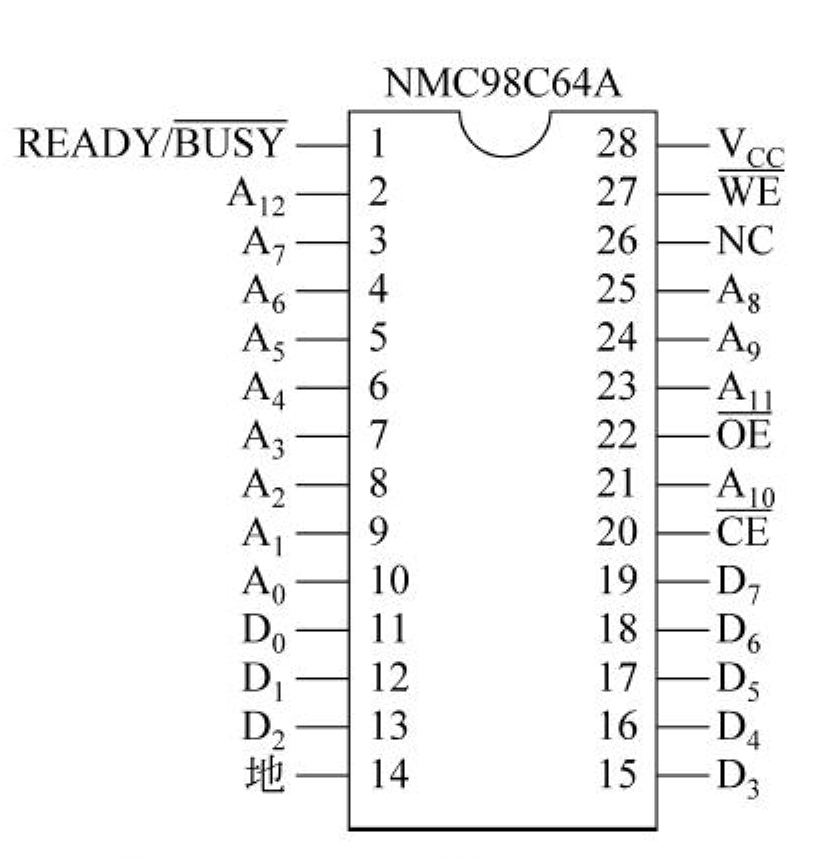


图 5-29　NMC98C64A 引线图

（6）READY/$\overline{BUSY}$是状态输出端。98C64A 正在执行编程写入时，此管脚为低电平；写完后，此管脚变为高电平。因为正在写入当前数据时，98C64A 不接收 CPU 送来的下一个数据，所以 CPU 可以通过检查此引脚的状态来判断写操作是否结束。

2. 98C64A 的工作过程

98C64A 的工作过程同样包括 3 个部分，即数据读出、编程写入和擦除。

1）数据读出

从 EEPROM 读出数据的过程与从 EPROM 及 RAM 中读出数据的过程一样。当$\overline{CE}$＝0，$\overline{OE}$＝0，$\overline{WE}$＝1 时，只要满足芯片所要求的读出时序关系，则可从选中的存储单元中将数据读出。

2）编程写入

将编程写入 98C64A 有两种方式：字节写入和自动页写入。

（1）字节写入。字节写入方式是一次写入一个字节的数据。但写完一个字节之后并不能立刻写下一个字节，而是要等到 READY/$\overline{BUSY}$端的状态由低电平变为高电平后才能开始下一个字节的写入。这是 EEPROM 芯片与 RAM 芯片在数据写入上的一个很重要的区别。

不同的芯片写入一个字节所需的时间略有不同，一般是几到几十毫秒。98C64A 需要的时间一般为 5ms，最大是 10ms。在对 EEPROM 编程时，可以通过查询 READY/

$\overline{BUSY}$引脚的状态来判断是否写完一个字节，也可利用该引脚的状态产生中断请求来通知 CPU 已写完一个字节。对于没有 READY/$\overline{BUSY}$信号的芯片，则可用软件或硬件定时的方式（定时时间应大于等于芯片的写入时间），以保证数据的可靠写入。当然，这种方法虽然在原理上比较简单，但会降低 CPU 的效率。

98C64A 的编程时序图如图 5-30 所示。从图中可以看出，当$\overline{CE}=0$，$\overline{OE}=1$ 时，只要在$\overline{WE}$端加上 100ns 的负脉冲，便可以将数据写入指定的地址单元。

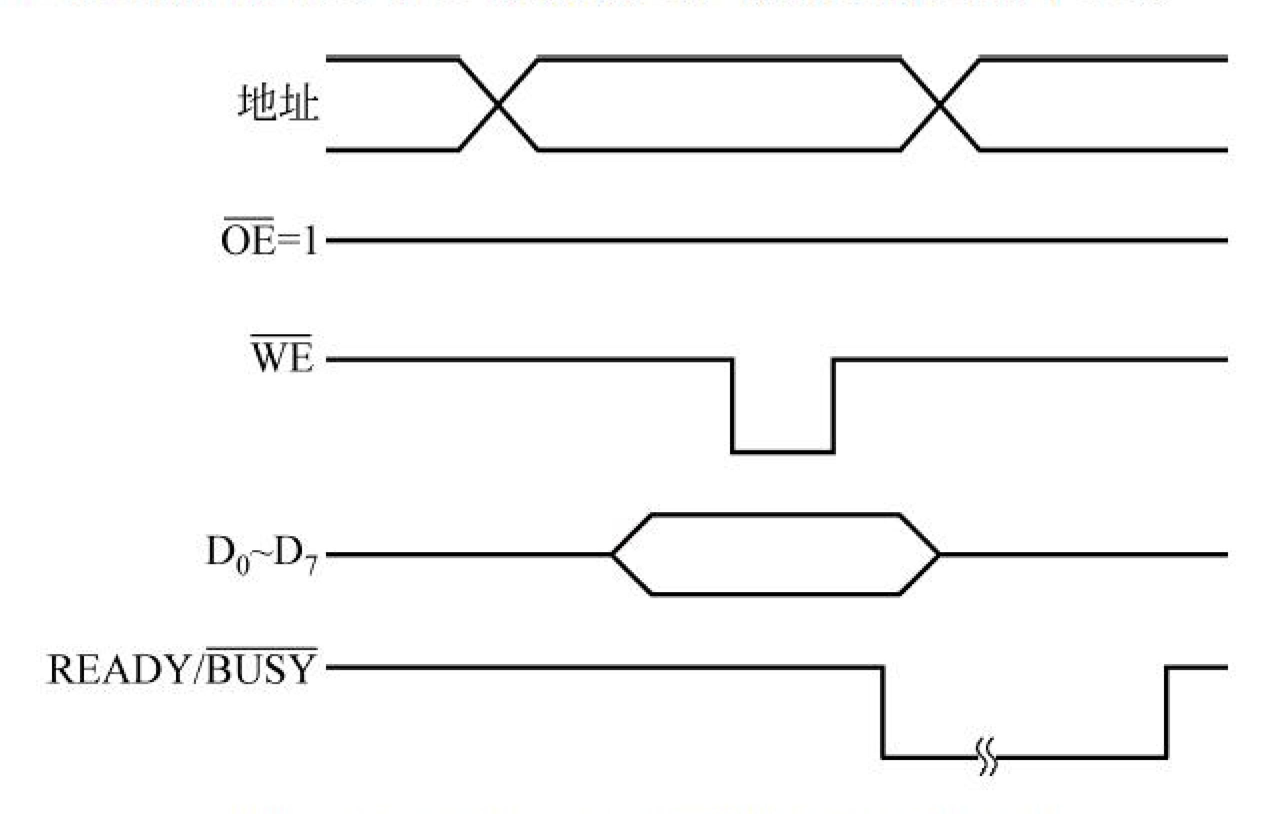


图 5-30　NMC98C64A 编程写入时序图

（2）自动页写入。页编程的基本思想是一次写完一页，而不是只写一个字节。每写完一页判断一次 READY/$\overline{BUSY}$端的状态。在 98C64A 中，一页数据为 1～32 个字节，要求这些数据在内存中是连续排列的。98C64A 的高位地址线 A_{12}～A_5 用来决定访问哪一页数据，低位地址线 A_4～A_0 用来决定寻址一页内所包含的 32 个字节。因此将 A_{12}～A_5 称为页地址。

其写入的过程是：利用软件首先向 EEPROM 98C64A 写入页的一个数据，并在此后的 300μs 内连续写入本页的其他数据，再利用查询或中断检查 READY/$\overline{BUSY}$端的状态是否已变高，若变高，则表示这一页的数据已写结束，然后接着开始写下一页，直到将数据全部写完。利用此方法，对 8K×8b 的 98C64A 来说，写满该芯片只需 2.6 秒。

3）擦除

擦除和写入是同一种操作，只不过擦除总是向单元中写入“FFH”而已。EEPROM 的特点是一次既可擦除一个字节，也可擦除整个芯片的内容。如果需要擦除一个字节，其过程与写入一个字节的过程完全相同，写入数据 FFH，就等于擦除了这个单元的内容。若希望一次将芯片所有单元的内容全部擦除干净，可利用 EEPROM 的片擦除功能，即在 D_0～D_7 上加上 FFH，使$\overline{CE}=0$，$\overline{WE}=0$，并在$\overline{OE}$引脚上加上＋15V 电压，使这种状态保持 10ms，就可将芯片所有单元擦除干净。

EEPROM 98C64A 有写保护电路，加电和断电不会影响芯片的内容。写入的内容一般可保存 10 年以上。每一个存储单元允许擦除/编程上万次。

3. EEPROM 的应用

EEPROM 可以很方便地实现与微机系统的连接，并可通过软件完成数据的读写。

这里要注意，尽管 EEPROM 可以实现在线读写，但绝不等于它可以像 RAM 芯片那样随机读写，对它的写入是有条件的，只有当 READY/$\overline{\text{BUSY}}$端的状态为高电平时才可以写入一个或一页数据。在 EEPROM 的应用中，如果需要读芯片某一单元的内容，只需执行一条存储器读指令就可将存储的数据读出；如果需要对 EEPROM 的内容重新编程，可以在连线状态下直接用字节或页方式写入。下面通过一个例子来说明 EEPROM 芯片的应用。

【例 5-5】 将一片 98C64A 接到系统总线上，使其地址范围为 3E000H～3FFFFH，并编程序将芯片的所有存储单元写入 66H。

题目分析：

根据 98C64A 芯片的特性，在对其进行写操作时，需首先判断 READY/$\overline{\text{BUSY}}$端的状态。该端状态需通过输入接口连接到系统的数据总线，当其为高电平时，可写入一次数据；该端为低电平则需等待。系统可以通过以下 3 种方式确定是否可对芯片进行写操作。

(1) 通过延时等待方式写入数据。可根据芯片工作时序所给出的参数，确定完成一次写操作所需要的时间。

(2) 通过查询 READY/$\overline{\text{BUSY}}$端的状态，判断一个写周期是否结束。

(3) 采用中断方式。可将 READY/$\overline{\text{BUSY}}$信号通过中断控制器连接到 CPU 的外部可屏蔽中断请求输入端，当 READY/$\overline{\text{BUSY}}$端由低电平(“忙”状态)变为高电平时，产生有效的 INTR 中断请求，CPU 响应中断后，向芯片进行一次写操作。

以下给出第(1)和第(2)种方式下对芯片进行写操作的程序。

设计电路连接如图 5-31 所示。READY/$\overline{\text{BUSY}}$端的状态通过一个接口电路送到

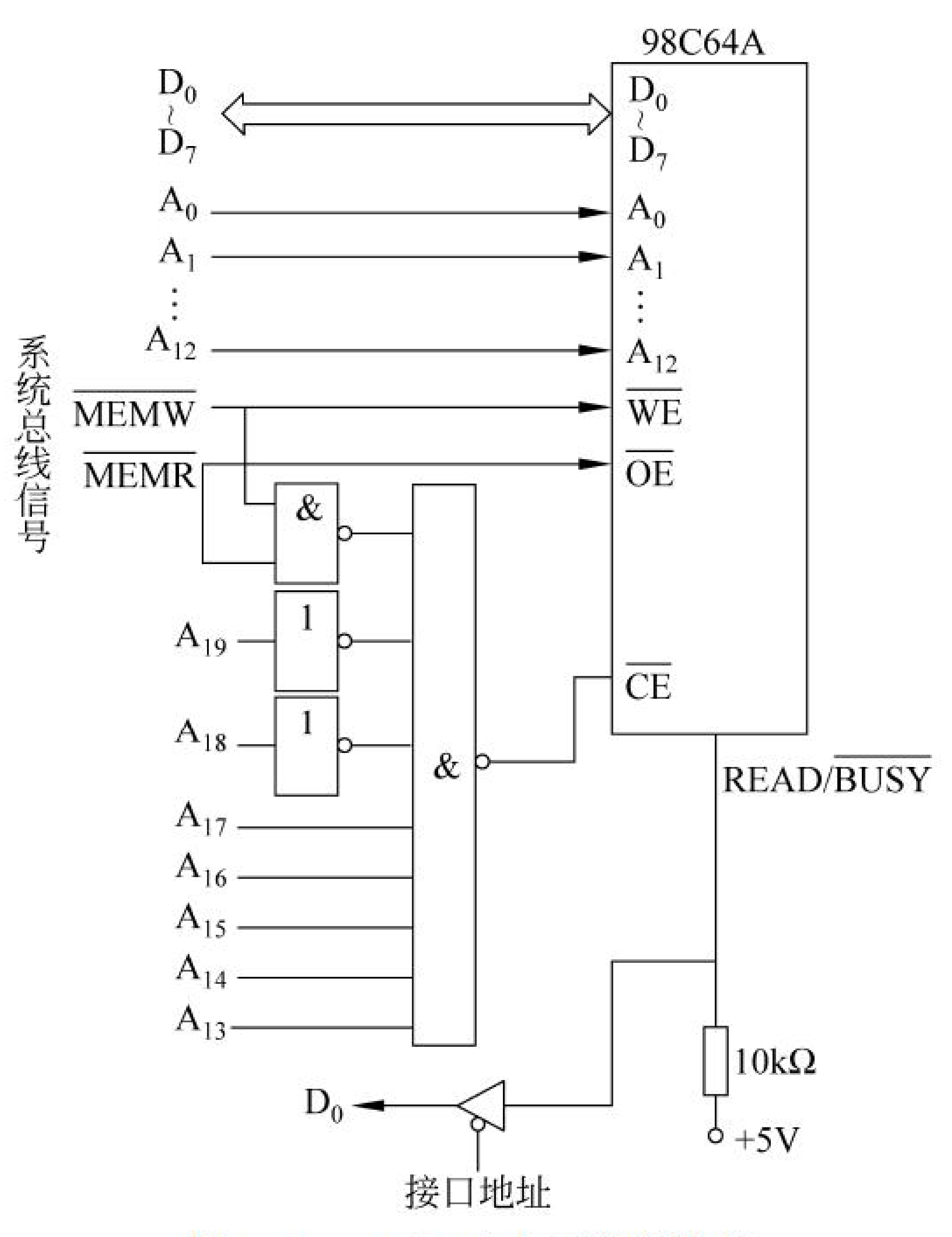


图 5-31　98C64A 与系统的连接

CPU 数据总线的 D_0 端,CPU 读入该状态以判断一个写周期是否结束。READY/$\overline{BUSY}$ 状态接口地址为 02E0H。

程序 1：用延时等待方式

```
START: MOV AX,3E00H
       MOV DS,AX                  ;段地址送 DS
       MOV SI,0000H               ;第一个单元的偏移地址送 SI
       MOV CX,2000H               ;芯片的存储单元个数送 CX
AGAIN: MOV AL,66H
       MOV [SI],AL                ;写入一个字节
       CALL TDELAY 120μs          ;调用延时子程序,延时 120μs
       INC SI                     ;下一个存储单元地址
       LOOP AGAIN                 ;若未写完则再写下一个字节
       HLT
```

程序 2：用查询 READY/$\overline{BUSY}$端状态的方式

```
START: MOV AX,3E00H
       MOV DS,AX                  ;段地址送 DS
       MOV SI,0000H               ;第一个单元的偏移地址送 SI
       MOV CX,2000H               ;芯片的存储单元个数送 CX
       MOV BL,66H                 ;要写入的数据送 BL
AGAIN: MOV DX,02E0H               ;READY/BUSY状态接口地址送 DX
WAIT:  IN AL,DX                   ;从接口读入 READY/BUSY端的状态
       TEST AL,01H                ;可以写入吗?
       JZ WAIT                    ;若为低电平(表示“忙”)则等待
       MOV [SI],BL                ;否则,写入一个字节
       INC SI                     ;下一个存储单元地址
       LOOP AGAIN                 ;若未写完则再写下一个字节
       HLT
```

5.3.3 闪存 FLASH

尽管 EEPROM 能够在线编程,而且可以自动页写入,使其在使用方便性及写入速度两个方面都较 EPROM 更进一步,但即便如此,其编程时间相对 RAM 而言还是太长,特别是对大容量的芯片更是如此。人们希望有一种写入速度类似于 RAM,掉电后内容又不丢失的存储器。为此,一种新型的称为闪存的 EEPROM 被研制出来。闪存的编程速度快,掉电后内容又不丢失,从而得到很广泛的应用。下面以 TMS28F040 芯片为例简单介绍闪存的工作原理。

1. 引线及结构

28F040 的外部引线如图 5-32 所示。它共有 19 根地址线和 8 根数据线，说明该芯片的容量为 512K×8b；$\overline{G}$ 为输出允许信号，低电平有效；$\overline{E}$ 是芯片写允许信号，在它的下降沿锁存选中单元的地址，用上升沿锁存写入的数据。

28F040 芯片将其 512KB 的容量分成 16 个 32KB 的块(或页)，每一块均可独立进行擦除。

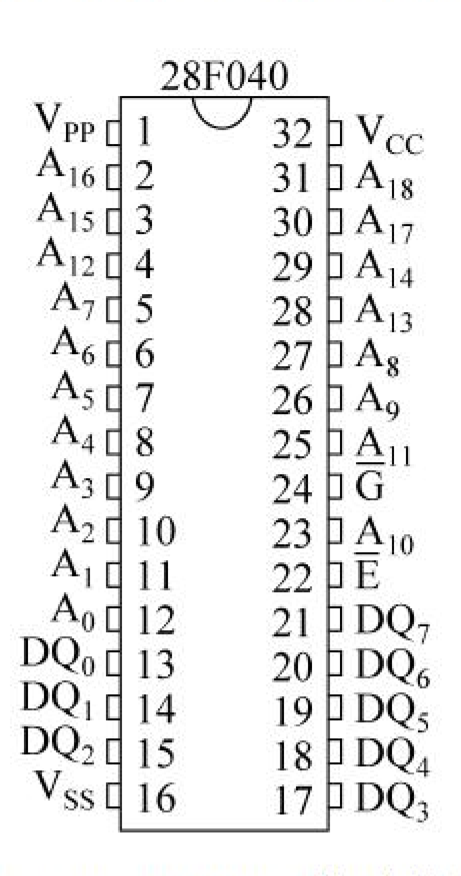


图 5-32　28F040 的引线图

2. 工作过程

28F040 与普通 EEPROM 芯片一样也有 3 种工作方式，即数据读出、编程写入和擦除。但不同的是它是通过向内部状态寄存器写入命令的方法来控制芯片的工作方式，对芯片所有的操作都要先向状态寄存器写入命令。另外，28F040 的许多功能需要根据状态寄存器的状态来决定。要知道芯片当前的工作状态，只需写入命令 70H 就可读出状态寄存器各位的状态了。状态寄存器各位的含义和 28F040 的命令分别如表 5-2 和表 5-3 所示。

表 5-2　状态寄存器各位的含义

位	高电平(1)	低电平(0)	用于
$SR_7(D_7)$	准备好	忙	写命令
$SR_6(D_6)$	擦除挂起	正在擦除/已完成	擦除挂起
$SR_5(D_5)$	块或片擦除错误	片或块擦除成功	擦除
$SR_4(D_4)$	字节编程错误	字节编程成功	编程状态
$SR_3(D_3)$	V_{PP} 太低，操作失败	V_{PP} 合适	监测 V_{PP}
$SR_2 \sim SR_0$			保留未用

表 5-3　28F040 的命令

命　令	总线周期	第一个总线周期			第二个总线周期		
		操作	地址	数据	操作	地址	数据
读存储单元	1	写	×	00H			
读存储单元	1	写	×	FFH			
读标记	3	写	×	90H	读	IA(1)	
读状态寄存器	2	写	×	70H	读	×	SRD(4)
清除状态寄存器	1	写	×	50H			
自动块擦除	2	写	×	20H	写	BA(2)	D0H
擦除挂起	1	写	×	B0H			
擦除恢复	1	写	×	D0H			

续表

命　　令	总线周期	第一个总线周期			第二个总线周期		
		操作	地址	数据	操作	地址	数据
自动字节编程	2	写	×	10H	写	PA(3)	PD(5)
自动片擦除	2	写	×	30H	写		30H
软件保护	2	写		0FH	写	BA(2)	PC(6)

注：

(1) 若是读厂家标记，IA＝00000H，读器件标记则 IA＝00001H；

(2) BA 为要擦除块的地址；

(3) PA 为欲编程存储单元的地址；

(4) SRD 是由状态寄存器读出的数据；

(5) PD 为要写入 PA 单元的数据；

(6) PC 为保护命令，若 PC＝00H——清除所有的保护，PC＝FFH——置全片保护，PC＝F0H——清地址指定的块保护，PC＝0FH——置地址指定的块保护。

1）数据读出

数据读出包括读出芯片中某个单元的内容、读出内部状态寄存器的内容以及读出芯片内部的厂家及器件标记 3 种情况。如果要读某个存储单元的内容，则在初始加电以后或在写入命令 00H（或 FFH）之后，芯片就处于只读存储单元的状态。这时就和读 SRAM 或 EPROM 芯片一样，很容易读出指定的地址单元中的数据。此时的 V_{PP}（编程高电压端）可与 V_{CC}（＋5V）相连。

2）编程写入

编程写入方式包括对芯片单元的写入和对其内部每个 32KB 块的软件保护。软件保护是用命令使芯片的某一块或某些块规定为写保护，也可置整片为写保护状态，这样可以使被保护的块不被写入新的内容或擦除。例如，向状态寄存器写入命令 0FH，再送上要保护的块的地址，就可置规定的块为写保护；若写入命令 FFH，就置全片为写保护状态。

28F040 对芯片的编程写入采用字节编程方式，其写入过程如图 5-33 所示。

首先，28F040 向状态寄存器写入命令 10H，再在指定的地址单元写入相应数据；接着查询状态，判断这个字节是否写好；写好则重复这个过程，直到全部字节写入完毕。这个过程与前面介绍的 98C64 的字节编程类似。98C64 是由 READY/$\overline{\text{BUSY}}$端的状态来指示其是否允许写下一个字节，而 28F040 则以状态寄存器的状态来指示其是否允许写下一个字节。

28F040 的编程速度很快，其一个字节的写入时间仅为 8.6μs。

3）擦除

28F040 既可以每次擦除一个字节，也可以一次擦除整个芯片，或根据需要只擦除片内某些块，并可在擦除过程中使擦除挂起和恢复擦除。

对字节的擦除，实际上就是在字节编程过程中，写入数据的同时就等于擦除了原单元的内容；对整片擦除，擦除的标志是擦除后各单元的内容均为 FFH。整片擦除最快只需 2.6s，但受保护的内容不被擦除，也允许对 28F040 的某一块或某些块擦除，每 32KB 为一块，块地址由 A_{15}～A_{18}来决定。在擦除时，只要给出该块的任意一个地址（实际上只关心 A_{15}～A_{18}）即可。整片擦除及块擦除的流程图分别如图 5-34 中的(a)和(b)所示。擦除一

块的最短时间为 100ms。

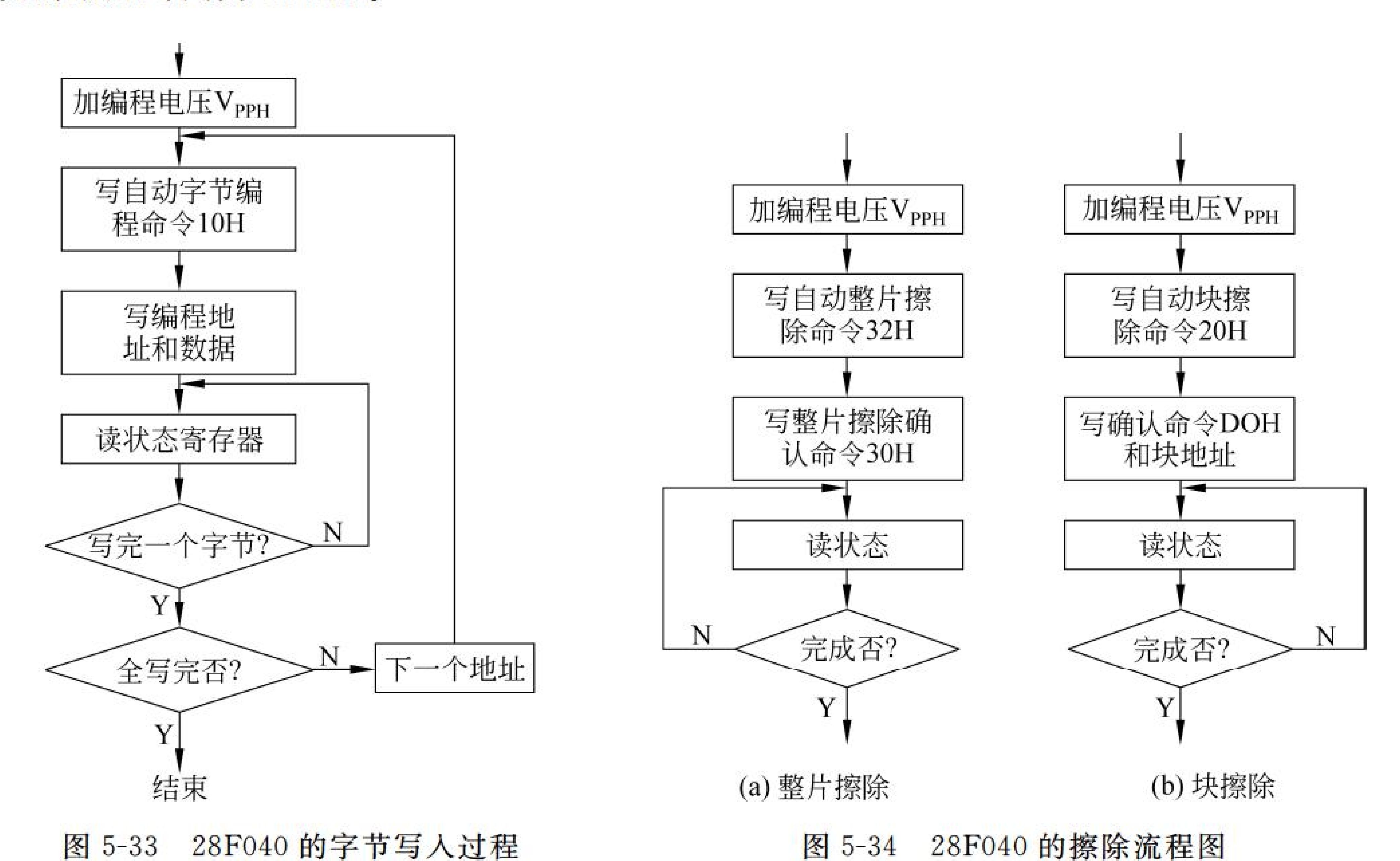


图 5-33　28F040 的字节写入过程　　　　图 5-34　28F040 的擦除流程图

擦除挂起是指在擦除过程中需要读数据时，可以利用命令暂时挂起擦除，读完后又可用命令恢复擦除。

28F040 在使用中，要求在其引线控制端加上适当电平，以保证芯片正常工作。不同工作类型的 28F040 的工作条件是不一样的，具体如表 5-4 所示。

表 5-4　28F040 的工作条件

	E	G	V_{PP}	A_9	A_0	D_0～D_9
只读存储单元	V_{IL}	V_{IL}	V_{PPL}	×	×	数据输出
读	V_{IL}	V_{IL}	×	×	×	数据输出
禁止输出	V_{IL}	V_{IH}	V_{PPL}	×	×	高阻
准备状态	V_{IH}	×	×	×	×	高阻
厂家标记	V_{IL}	V_{IL}	×	V_{ID}	V_{IL}	97H
芯片标记	V_{IL}	V_{IL}	×	V_{ID}	V_{IH}	79H
写入	V_{IL}	V_{IH}	V_{PPH}	×	×	数据写入

注：V_{IL}为低电平，V_{IH}为高电平 V_{CC}，V_{PPL}为 0～V_{CC}，V_{PPH}为＋12V，V_{ID}为＋12V，×表示高低电平均可。

3. 闪存的应用

目前闪存主要用来构成存储卡，以代替软磁盘。存储卡的容量可以做得较软盘大，但具有软盘的方便性，现在已大量用于便携式计算机、数码相机、MP3 播放器等设备中。

另外，闪速 EEPROM 也用作内存，用于存放程序或不经常改变且对写入时间要求不高的场合，如微机的 BIOS、显卡的 BIOS 等。

5.4 高速缓冲存储器

一个微机系统整体性能的高低与许多因素有关,如 CPU 主频的高低、存储器的存取速度、系统架构、指令结构、信息在各部件之间的传送速度等,而 CPU 与内存之间的存取速度则是一个很重要的因素。如果只是 CPU 工作速度很高,但内存存取速度较低,就会造成 CPU 经常处于等待状态,既降低了处理速度,又浪费了 CPU 的能力。例如,主频为 733MHz 的 Pentium Ⅲ一次指令执行时间为 1.35ns,与其相配的内存(SDRAM)存取时间为 7ns,比前者慢 5 倍,二者速度相差很大。

减少 CPU 与内存之间速度差异的办法主要有 3 种:①在基本总线周期中插入若干等待周期,让 CPU 等待内存的数据,这样做虽然方法简单,但显然会浪费 CPU 的能力;②采用存取速度较快的 SRAM 作存储器,这样虽可基本解决 CPU 与存储器之间速度不匹配的问题,但成本很高,而且 SRAM 的速度始终不能赶上 CPU 速度的发展;③在慢速的 DRAM 和快速的 CPU 之间插入一个速度较快、容量较小的 SRAM,起到缓冲作用,使 CPU 既可以以较快速度存取 SRAM 中的数据,又不使系统成本上升过高,这就是高速缓冲存储器(Cache),如图 5-35 所示。目前的微型机系统中一般均采用这种方法来提高存储系统的性能,使系统在成本增加不高的情况下,性能有较显著的提升。

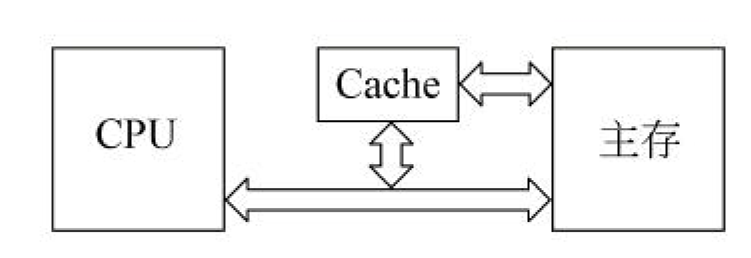


图 5-35　Cache 在微机系统中的位置

本节将简单介绍 Cache 的概念、原理、结构设计以及在微型机和 CPU 中的实现。

5.4.1　Cache 的工作原理

Cache 的工作原理是基于程序和数据访问的局部性。

任何程序或数据要为 CPU 所使用,必须先放到主存储器,即内存中。CPU 只与主存交换数据,所以主存的速度在很大程度上决定了系统的运行速度。对大量典型程序运行情况的分析结果表明,程序运行期间,在一个较短的时间间隔内,由程序产生的内存访问地址往往集中在存储器的一个很小范围的地址空间内。这一点其实很容易理解。指令地址本来就是连续分布的,再加上循环程序段和子程序段要多次重复执行。因此,对这些地址中的内容的访问就自然具有时间上集中分布的倾向。数据分布的这种集中倾向不如指令明显,但对数组的存储和访问以及内存变量的安排都使存储器地址相对集中。这种在单位时间内对局部范围的存储器地址频繁访问,而对此范围以外的地址则访问甚少的现象被称为程序访问的局部化(Locality of Reference)性质或程序访问的局部性。

由此可以想到,如果把在一段时间内一定地址范围中被频繁访问的信息集合,成批地从主存读到一个能高速存取的小容量存储器中存放起来,供程序在这段时间内随时使用,从而减少或不再去访问速度较慢的主存,就可以加快程序的运行速度。这就是 Cache 的设计思想,在 CPU 和主存之间设置一个小容量的高速存储器,称为高速缓冲存储器。不

难看出,程序和数据访问的局部化性质是 Cache 得以实现的原理基础。

有了 Cache,系统在工作时就总是不断地将与当前指令集相关联的一个不太大的后继指令集合从内存读到高速 Cache,然后再与 CPU 高速传送,从而达到速度匹配。CPU 在读取指令或数据时,总是先在 Cache 中寻找,若找到便直接读入 CPU,这称为"命中";找不到再到主存中查找,称为"未命中"。当 CPU 访问主存读取"未命中"的指令和数据时,将把这些信息同时写入 Cache 中,以保证下次命中。所以在程序执行过程中,Cache 的内容总是在不断地更新。

由于局部性原理不能保证所请求的数据 100%在 Cache 中,这里便存在一个命中率问题。所谓命中率就是在 CPU 访问 Cache 时,所需信息恰好在 Cache 中的概率。命中率越高,正确获取数据的可能性就越大。如果高速缓存的命中率为 92%,可以理解为 CPU 在访问存储器时,用 92%的时间与 Cache 交换数据,8%的时间与主存交换数据。

一般来说,Cache 的存储容量比主存的容量小得多,但不能太小,太小会使命中率太低;但也没有必要过大,过大不仅会增加成本,而且当 Cache 容量超过一定值后,命中率随容量的增加将不会有明显的增长。所以,Cache 的空间与主存空间在一定范围内应保持适当比例的映射关系,以保证 Cache 有较高的命中率,并且系统成本不过大地增加。一般情况下,可以使 Cache 与内存的空间比为 1∶128,即 256KB 的 Cache 可映射 32MB 内存;512KB 的 Cache 可映射 64MB 内存。在这种情况下,命中率都在 90%以上,即 CPU 在运行程序的过程中,有 90%的指令和数据可以在 Cache 中取得,只有 10%需要访问主存。对没有命中的数据,CPU 只好直接从内存获取,获取的同时也把它复制到 Cache 中,以备下次访问。

Cache 的命中率与 Cache 的大小、替换算法、程序特性等因素有关。假设 Cache 的命中率为 H,存取时间为 T_1,主存的存取时间为 T_2,则 Cache 存储器系统的平均存取时间 T 可用式(5.6)计算

$$T = T_1 \times H + T_2 \times (1 - H) \tag{5.6}$$

【例 5-6】 某微型机存储器系统由一级 Cache 和 RAM 组成。已知 RAM 的存取时间为 80ns,Cache 的存取时间为 6ns,Cache 的命中率为 85%,求该存储器系统的平均存取时间。

解: 由式(5.6)得

系统的平均存取时间=6ns×85%+80ns×15%=5.1ns+12ns=17.1ns

可以看出,有了 Cache 以后,CPU 访问主存的速度大大提高了。但要注意的是,增加 Cache 只是加快了 CPU 访问存储器系统的速度,而 CPU 访问存储器系统仅是计算机全部操作的一部分,所以增加 Cache 对系统整体速度只能提高 10%~20%。另外,若访问 Cache 没有命中的话,CPU 还要访问主存,这时反而延长了存取时间。所以按式(5.6)计算出来的平均存取时间仅是一个粗略值。

5.4.2 Cache 的读写操作

由于处理器需要主存储器的速度与主存储器实际具有的存取速度之间存在一个数量

级的差距，为了弥补这一差异，提高整个系统的性能，引入了 Cache 技术。Cache 是在逻辑上位于处理器与主存之间的部件，是内存储器的一部分。因此，对 Cache 的操作也包括读和写两种。

1. 贯穿读出法

贯穿读出法(Look Through)的原理示意图如图 5-36 所示。

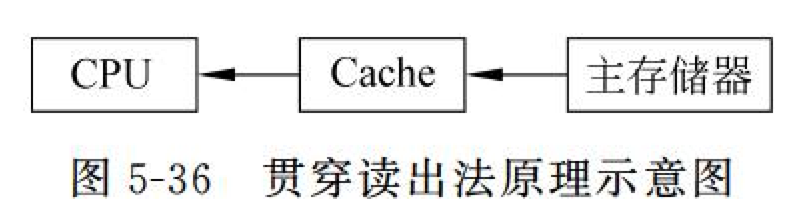


图 5-36　贯穿读出法原理示意图

在这种方式下，Cache 隔在 CPU 与主存之间，CPU 对主存的所有数据请求都首先送到 Cache，由 Cache 自行在自身查找。如果命中，则切断 CPU 对主存的请求，并将数据送出；如果不命中，则将数据请求传给主存。该方法的优点是降低了 CPU 对主存的请求次数，缺点是延迟了 CPU 对主存的访问时间。

2. 旁路读出法

旁路读出法(Look Aside)的原理示意图如图 5-37 所示。

在这种方式中，CPU 发出数据请求时，并不是单通道地穿过 Cache，而是向 Cache 和主存同时发出请求。由于 Cache 速度更快，如果命中，则 Cache 在将数据回送给 CPU 的同时还来得及中断 CPU 对主存的请求；若不命中，则 Cache 不做任何动作，由 CPU 直接访问主存。它的优点是没有时间延迟；缺点是每次 CPU 都要访问主存，占用了部分总线时间。



图 5-37　旁路读出法原理示意图

图 5-38　写直达法原理示意图

3. 写直达法

任一从 CPU 发出的写信号送到 Cache 的同时，也写入主存，以保证主存的数据能同步更新。写直达法(Write Through)的优点是操作简单，但由于主存的慢速，降低了系统的写速度并占用了部分总线时间。写直达法的原理示意图如图 5-38 所示。

4. 回写法

为了克服写直达法中每次数据写入都要访问主存，从而导致系统写速度降低并占用总线时间的弊病，尽量减少对主存的访问次数，又有了回写法(Write Back)。回写法的原理示意图如图 5-39 所示。它的工作原理是这样的：数据一般只写到 Cache，而不写入主存，从而使写入的速度加快。

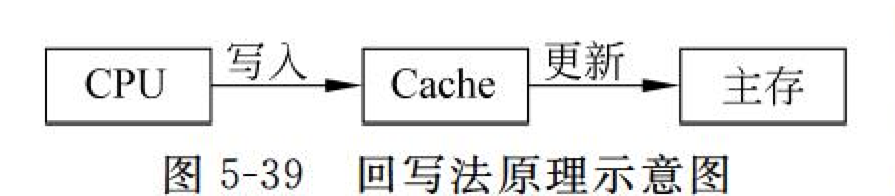


图 5-39　回写法原理示意图

5.4.3 Cache 与主存的存取一致性

由 5.1.1 节知，对 Cache 的管理全部是由硬件实现的，不论是应用程序员还是系统程序员，都看不到系统中有 Cache 存在，在他们的感觉中，程序是存放在主存中的。所以，在 Cache 存储器系统中，存储器的编址方式与主存储器是完全一致的。正常情况下，Cache 中存放的内容应该是主存的部分副本，即 Cache 中的内容应与主存对应地址中的内容相同。然而，由于以下两个原因，在一段时间内，主存某单元的内容和 Cache 对应单元中的内容可能会不相同，即造成了 Cache 中数据与主存储器中数据的不一致。

(1) 如图 5-40(a)所示，当 CPU 向 Cache 中写入一个数据时，Cache 某单元中的数据就从 X 被修改成了 X'，而主存对应单元中的内容则没有改，还是 X。

(2) 在输入输出操作中，I/O 设备的数据会写入到主存，修改了主存中的内容，将 X 变成了 X'，如图 5-40(b)所示，但 Cache 对应单元中的内容此时还是 X。

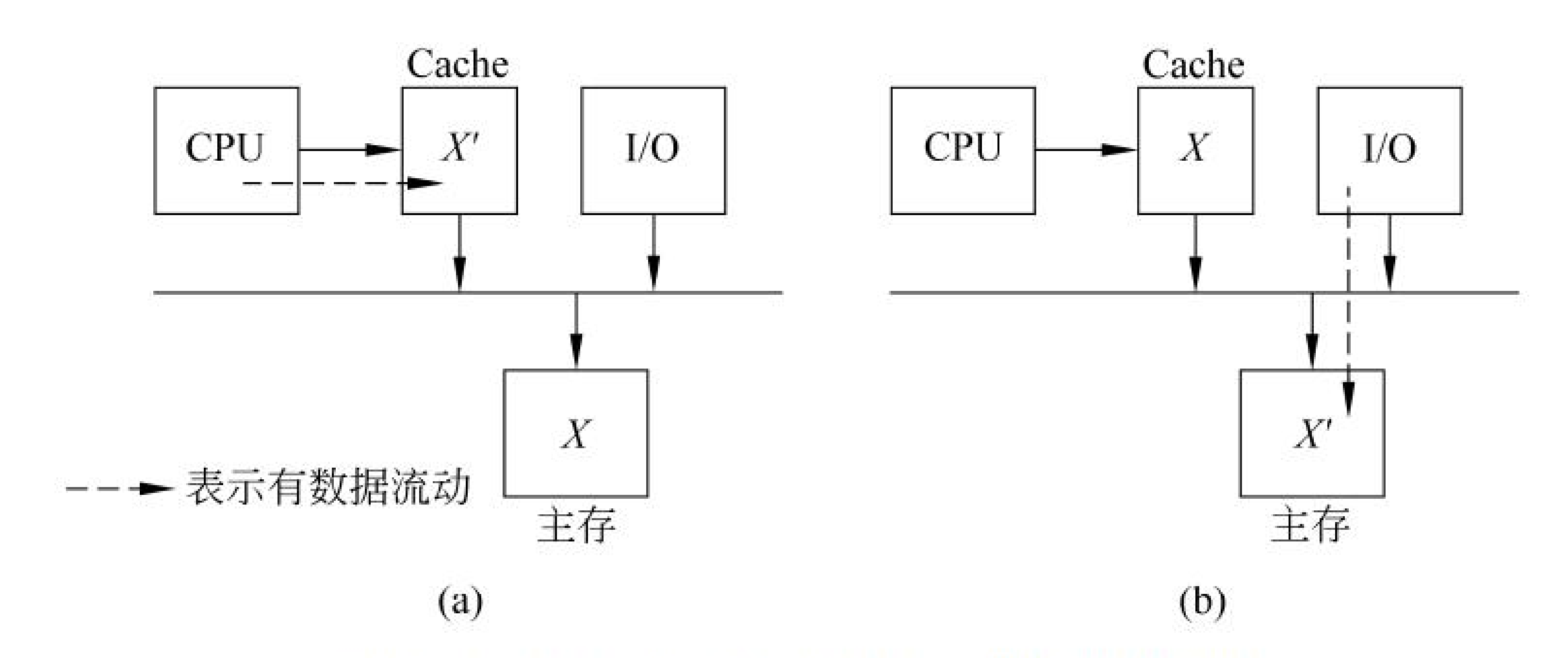


图 5-40　Cache 与主存数据不一致的两种情况

对第(1)种情况，如果此时要将主存中的包括 X 在内的数据输出到外设，则输出的是陈旧或错误的数据；对第(2)种情况，如果 CPU 读入了 Cache 中的数据 X，同样会造成错误。

为了避免 Cache 与主存储器中数据的不一致性，必须将 Cache 中的数据及时更新并准确地反映到主存储器。解决这个问题的方法就是在写操作时采用以上讲到的写直达法或回写法。

由于写直达法是在写 Cache 时，同时将数据写入主存，所以主存中的数据和 Cache 中的数据是一致的；对回写法，由于数据只写入 Cache 而不写入主存，就可能出现 Cache 中的数据得到更新，而对应主存中的数据却没有变(即数据不同步)的情况。因此，在采用回写方式时，可在 Cache 中设一个标志地址及数据陈旧的信息，只有当 Cache 中的数据被再次更改时，将原更新的数据写入主存相应的单元中，然后再接受再次更新的数据。这样保证了 Cache 和主存中数据的一致性。

5.4.4 Cache 的分级体系结构

一个微处理器的性能通常由如下几种因素估算：

$$性能 = \frac{kf}{CPI + (1 - H) \times N} \tag{5.7}$$

式中：k 为比例常数；f 为工作频率；CPI 为执行每条指令需要的周期数；H 为 Cache 的命中率；N 为存取周期数。

显然，为了提高处理器的性能，应尽量提高工作频率 f，减少执行每条指令需要的周期数 CPI，提高 Cache 的命中率 H，减少存取周期数 N。要达到这些目的，可采用以下技术。

(1) 同时分发多条指令和采用乱序执行，可以减少 CPI 的值。

(2) 采用转移预测和适当增加 Cache 容量，可以提高 H 的值。

(3) 采用高速的总线接口和不分块的 Cache 方案，可以减少存取周期数 N。

(4) 采用指令数据预取技术，可以提高 Cache 的命中率 H。

在现代微机系统中，仅采用一个级别的 Cache 还不能满足要求，而需要增加第二级 Cache 甚至三级 Cache，这就构成了 Cache 的分级结构。

1. 一级 Cache

在 Pentium 微处理器中，一级 Cache(L1 Cache)集成在 CPU 片内。为了减少 Cache 的冲突，L1 Cache 分为指令 Cache 和数据 Cache，使指令和数据的访问互不影响。指令 Cache 用于存放预取的指令，内部具有写保护功能，能够防止代码被无端破坏。

数据 Cache 中存放指令的操作数。为了保持数据的一致性，数据 Cache 中的每一个 Cache 行(进行一次 Cache 操作的数据位数，对 Pentium 微处理器，一个 Cache 行的宽度为 32B)都设置了 4 个状态，由这些状态定义一个 Cache 行是否有效，在系统的其他 Cache 中是否可用，是否为已修改状态等。这 4 个状态分别被称为 M(Modified)状态、E(Exclusive)状态、S(Shared)状态及 I(Invalid)状态。表 5-5 给出了 Cache 行在某一时刻各状态位的状态。

表 5-5 Cache 行状态

Cache 行状态	M(已修改)	E(独占)	S(共享)	I(无效)
该 Cache 行是否有效	是	是	是	是
存储器复制是有效还是过期	过期	有效	有效	
其他 Cache 中是否保存有备份	无	无	可能有	可能有
该 Cache 行是否写到总线上	不写到总线	不写到总线	写到总线并修改 Cache	直接写到总线

2. 二级 Cache

为了提高微机的整体性能，在 PentiumⅡ之后的微处理器芯片上都配置了二级 Cache (L2 Cache)，其工作频率与 CPU 内核的频率相同。亦即为了能够高速地向 CPU 提供其运行所需要的信息，微机中的 Cache 存储器系统实际上可以说由三级存储器构成，如图 5-41 所示。其中，L1 Cache 主要是用于提高存取速度，主存主要用于提供足够的存储容量，而 L2 Cache 则是速度和存储容量兼备。

在 Pentium 系列微处理器中，L2 Cache 不再分为指令 Cache 和数据 Cache，而是将两

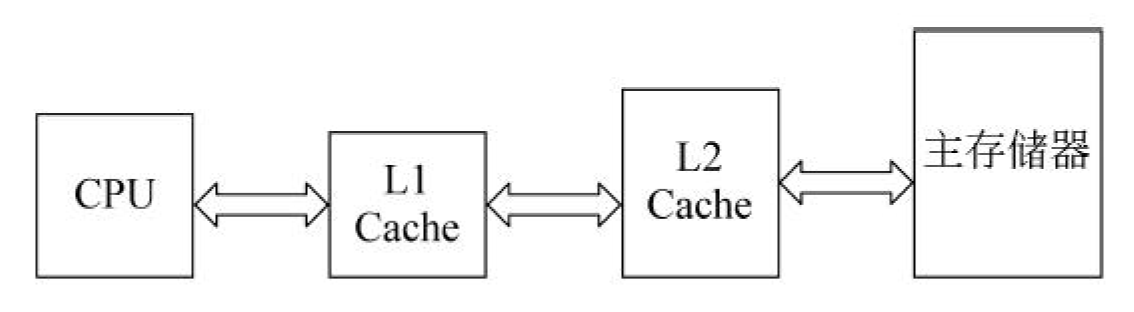


图 5-41　系统中的二级 Cache

者统一为一体。例如，当指令预取部件请求从指令 Cache 中预取指令时，如果命中，则直接读取；若不命中，L1 Cache 就会向 L2 Cache 发出预取请求，此时就会在 L2 Cache 中进行查找。如果找到(即命中)，就把找到的指令送一级指令 Cache(传送速度为每次 8B)；如果在 L2 Cache 中也不命中，则再向主存发出读取请求。

因此，L2 Cache 的存在使得当芯片内一级指令 Cache 和一级数据 Cache 出现不命中时，可以由 L2 Cache 提供处理器所需的指令和数据，而不必再去访问主存，从而提高了系统的整体性能。

对于一个有多级 Cache 的微型机系统，通常 80％的内存申请都可在一级缓存中实现，另外 20％的内存申请中的 80％又可只与二级缓存打交道。因此，只有 4％的内存申请定向到主存 DRAM 中。

L1 Cache 的容量为 8～64KB，L2 Cache 一般比 L1 Cache 大一个数量级以上，其容量为 128KB～2MB 不等。

Cache 分级结构的不足在于高速缓存组数目受限，需要占用线路板空间和一些支持逻辑电路，使成本增加。

随着计算机技术的发展，CPU 的主频已越做越高，系统构架越做越先进，而主存 DRAM 的结构和存取时间缩短的进程则相对较慢。因此 Cache 技术就愈显重要，结果使得在微机系统中的 Cache 越做越大。现在已把 Cache 的容量和速度作为评价和选购微机系统的一个重要指标。

5.5　半导体存储器设计举例

本节以部分具体应用实例进一步说明如何利用已有的存储器芯片设计出所需要的半导体存储器。举例之前，先根据本章所述内容，对半导体存储器设计给出以下几点说明。

(1) 任何存储芯片的存储容量都是有限的。单个芯片往往不能满足所需存储空间的要求，表现在芯片的存储单元个数不够或每单元的字长不够，或两者都不能满足要求。此时就需要用多个存储芯片进行组合。

计算机中的内存一般是都按字节来组织的，即每单元均存放 1B 数据，但实际的存储器芯片却并非每单元的字长都是 1B，它们可以是 1 位、4 位或 8 位的。如 5.2.3 中介绍的 DRAM 2164A 芯片，其容量为 64K×1b，即每单元的字长只有 1 位。此时，为了构成所需要的内存空间，须首先将每个单元的字长扩展到 8 位。这项工作称为“位扩展”。

若存储器芯片上每个存储单元的字长已满足要求(如字长已为 8 位)，而只是存储单元的个数不够，需要增加的是存储单元的数量。此时则需要增加单元数，这个工作相应地

就称为“字扩展”。

(2) 在“位扩展”构成的存储器系统中，每个单元中的内容都被存储在不同的存储器芯片上。因此，位扩展电路的连线方法是：每个存储芯片的地址线和控制线(包括片选信号线、读写信号线等)全部并联在一起，从而使各芯片具有同样的地址范围和同步的操作控制(这是“位扩展”必有的要求)，数据线分别引出至数据总线的不同位上。

(3) “字扩展”是对存储器容量的扩展，因此，系统中各个芯片必须要有不同的地址范围。故“字扩展”的连线特点是：每个芯片的地址信号、数据信号和读写信号等控制信号线按信号名称全部并联在一起，片选端分别引出到地址译码器的不同输出端，用以区别不同的芯片。

综上所述，存储器系统的设计可以分为以下几步。

(1) 根据现有芯片的类型及需求，确定所需要的芯片数量。

(2) 根据要求将芯片“多片并联”进行位扩展(如果需要的话)，设计出满足字长要求的“存储模块”；再对“存储模块”进行字扩展，构成符合要求的存储器，并确定相应的线路的连接方法。

(3) 设计译码电路。可根据不同需求，利用基本逻辑门或专用译码器完成相应译码电路的设计。

(4) 编写相应的存储器读写控制程序。

【例 5-7】 用 Intel 2164A 构成容量为 128KB 的内存。

题目分析：

由于 2164A 是 64K×1b 的芯片，所以首先要进行位扩展。用 8 片 2164A 组成 64KB 的内存模块，然后再用两组这样的模块进行字扩展。所需的芯片数为(128/64)×(8/1)=16 片。

要寻址 128K 个内存单元至少需要 17 位地址信号线($2^{17}=128K$)。而 2164A 有 64K 个单元，只需要 16 位地址信号(分为行和列)，余下的 1 根地址线用于区分两个 64KB 的存储模块，即构成此内存共需 16 片 2164 芯片；至少需要 17 根地址信号线，其中 16 根用于 2164 的片内寻址(行、列地址)，1 根用于片选地址译码。

由于 DRAM 芯片的外围控制线路比较复杂，本题参照图 5-18 所示的线路连接图，给出图 5-42 所示的示意图。

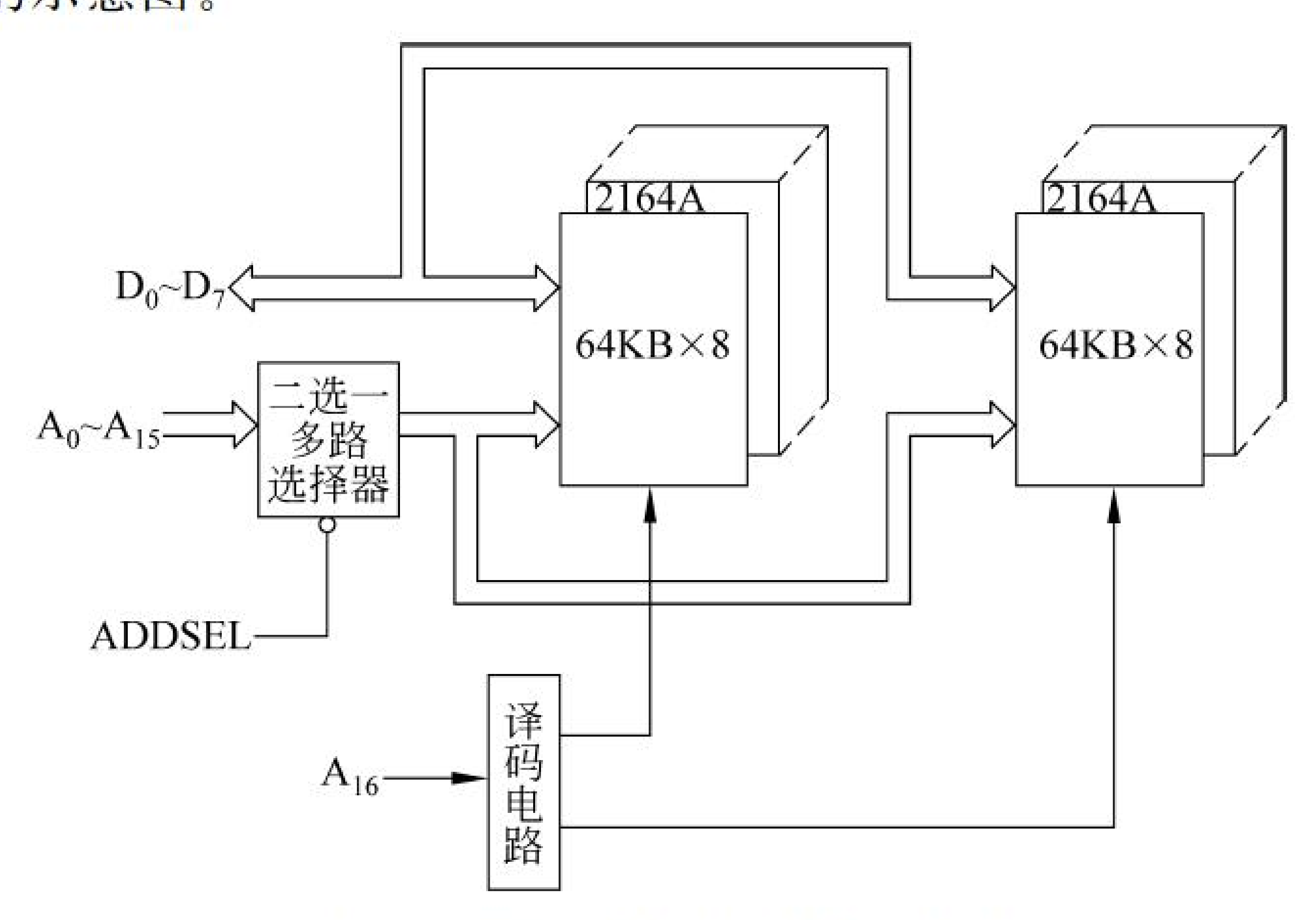


图 5-42　字/位扩展应用举例示意图

【例 5-8】 利用图 5-43 所示的 SRAM8256 存储器芯片(容量为 256K×8b)构成 1MB 的存储器。芯片各引线含义为:$A_0 \sim A_{17}$,地址线;$D_0 \sim D_7$,数据线;$\overline{WE}$,写允许信号线(低电平有效);$\overline{OE}$,读出允许信号(低电平有效);$\overline{CS}$,片选信号(低电平有效)。

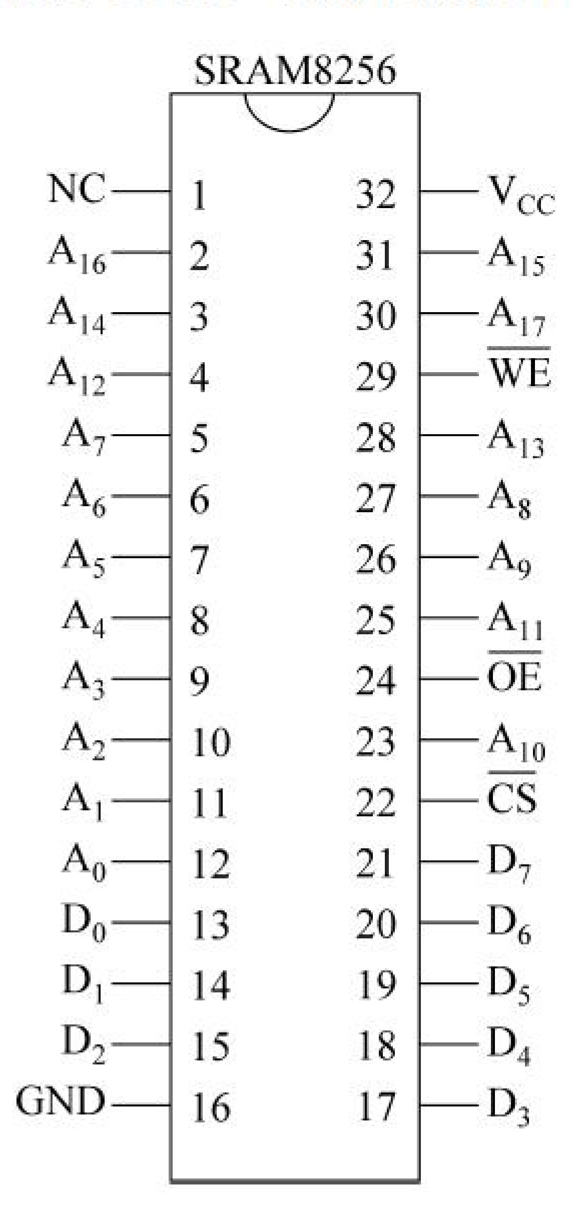


图 5-43 SRAM8256 引线

题目分析:

由于 SRAM8256 芯片的容量为 256KB,要构成 1MB 的存储器,需要 4 片芯片,4 片 8256 的地址范围分别为:00000H~3FFFFH、40000H~7FFFFH、80000H~BFFFFH、C0000H~FFFFFH。

这里仍然采用 72LS138 译码器构成译码电路。由于 SRAM8256 芯片有 18 根地址线,只有两根高位地址信号 A_{19} 和 A_{18} 可以用于片选译码,因此将 LS138 的输入端 C 直接接低电平,而使另外两个输入端 A 和 B 分别接到 A_{18} 和 A_{19},这两路高位地址信号的 4 种不同的组合分别选中 4 片 8256。

图 5-44 画出了存储器与系统的连接图。除片选信号外,其他所有的信号线都并联连接在系统总线上。

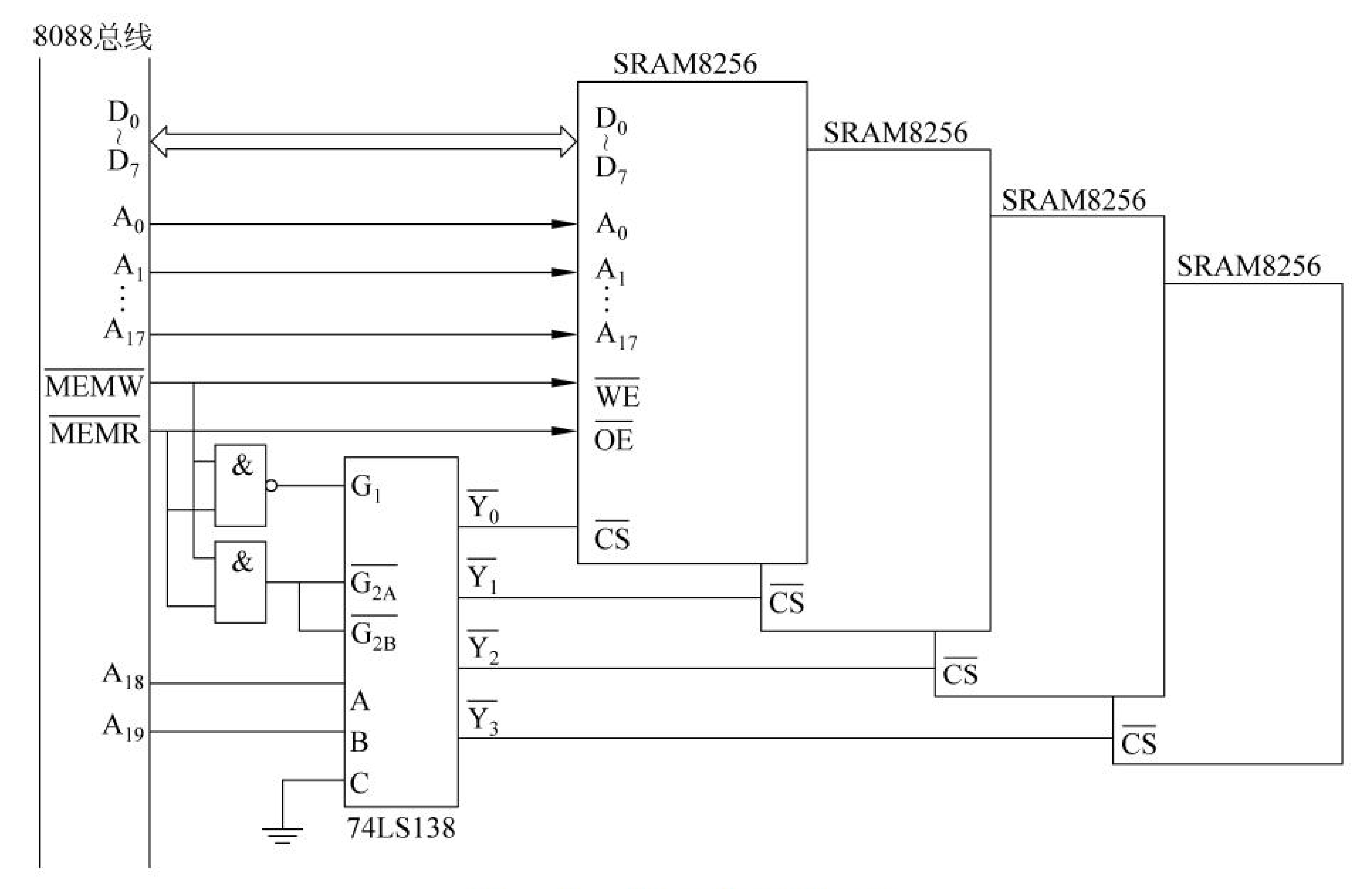


图 5-44 8256 的应用连接图

【例 5-9】 某 8088 系统使用 EPROM2764 和 SRAM6264 芯片组成 16KB 内存。其中:ROM 地址范围为 FE000H~FFFFFH,RAM 地址范围为 F0000H~F1FFFH。要求利用 74LS138 译码器设计译码电路,实现 16KB 存储器与系统的连接。

题目分析:

由 5.2.1 节和 5.3.1 节可知，SRAM6264 和 EPROM2764 芯片的存储容量均为 8KB，片内地址信号线 13 位，数据线 8 位。根据题目所给地址范围，得出芯片的高位地址，ROM：1111111，RAM：1111000。由此可设计出存储器与系统的接口电路如图 5-45 所示。

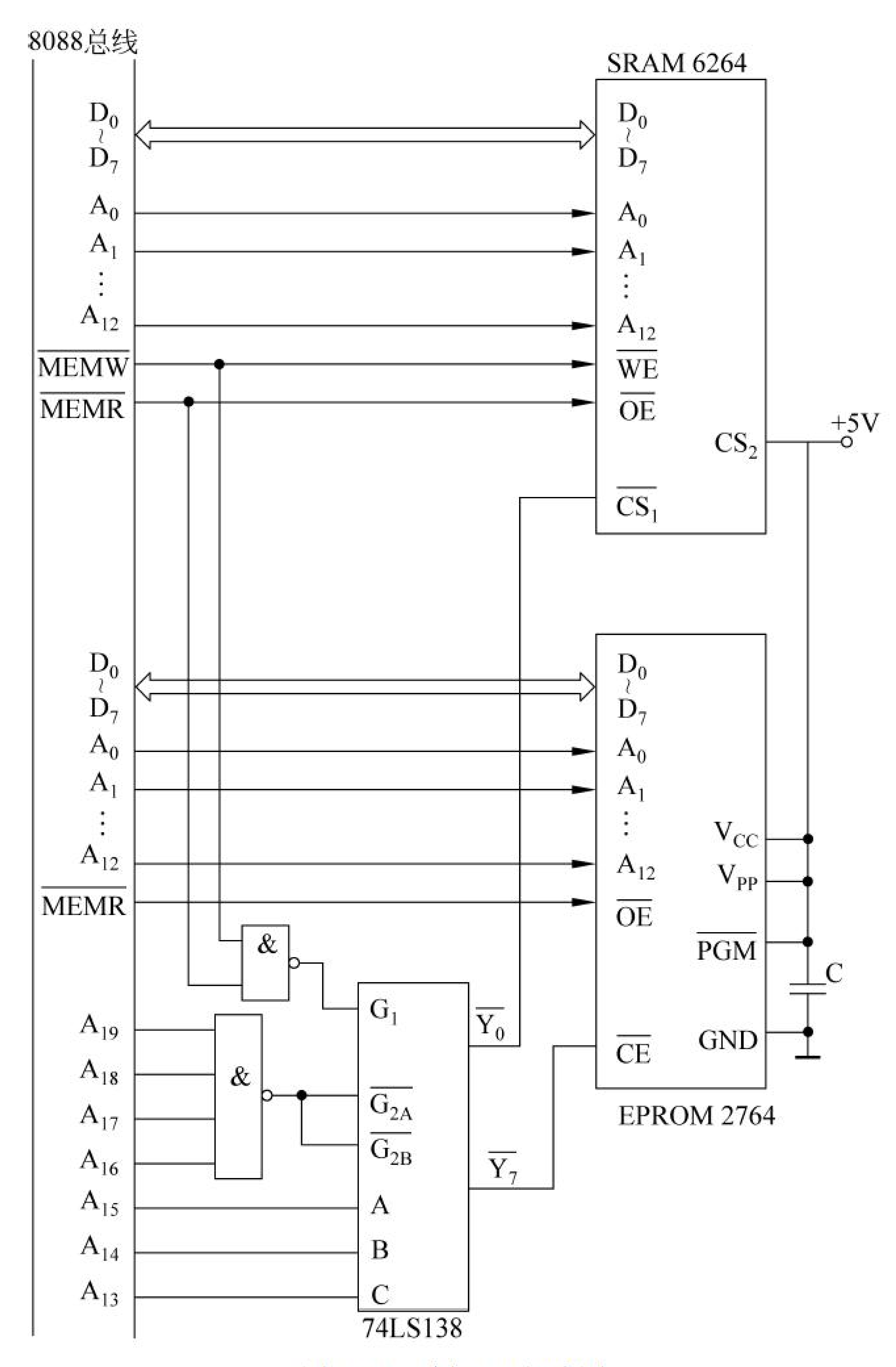


图 5-45　例 5-9 电路图

【例 5-10】 分别利用 SRAM6264 芯片和 EEPROM98C64A 芯片构造 32KB 的数据存储器及 32KB 的程序存储器，并将程序存储器各单元的初值置为 FFH。

要求数据存储器的地址范围为 90000H～97FFFH；程序存储器的地址范围为 98000H～9FFFFH；连接各 EEPROM 98C64A 的 READ/$\overline{BUSY}$端的接口地址为 380H～383H。

题目分析：

由于 6264 和 98C64A 芯片的存储容量均为 8KB，因此，根据题目要求，各需要 4 片芯片。

根据 EEPROM 芯片的特点，可利用其作为程序存储器。由题目要求，需对程序存储

器各单元置初值，其工作流程为：①地址总线上产生 20 位有效地址，其中，高 7 位地址信号用于选中对应的存储器芯片(即有效的$\overline{CE}$信号)，使其处于工作状态；②产生 16 位地址信号，同时使 IO/$\overline{M}$=1，且$\overline{RD}$=0，读取选中 EEPROM 芯片的 R/$\overline{B}$ 端状态；若 R/$\overline{B}$=1，则使 IO/$\overline{M}$=0，且$\overline{WR}$=0，并送上 20 位有效存储器单元地址，进行一次写操作。

设计系统如图 5-46 所示。

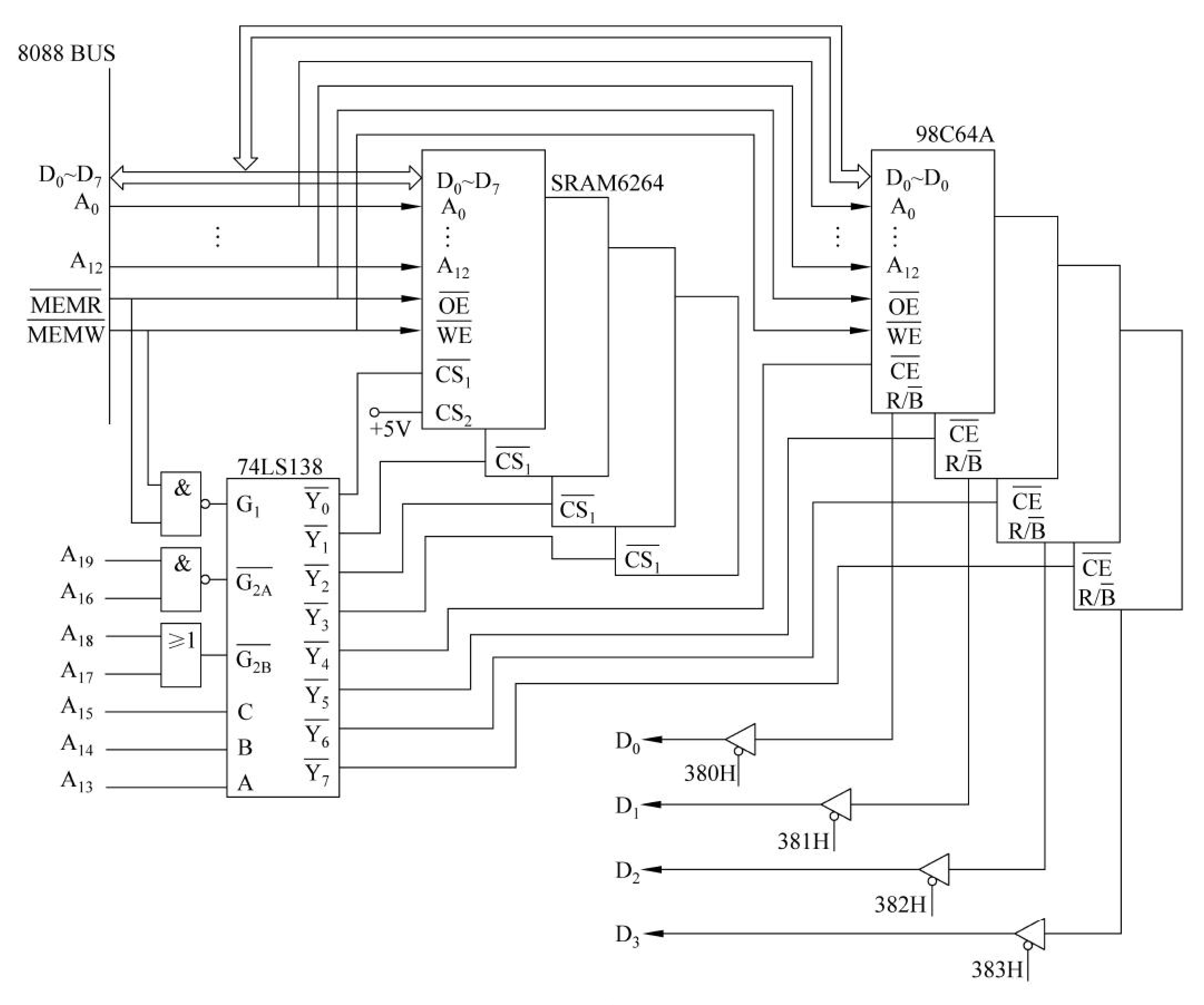


图 5-46　例 5-10 图

将程序存储器各单元置初值为 FFH 的程序段：

```
        MOV AX,9800H             ;设置段基地址
        MOV DS,AX
        MOV BX,104H              ;使 BH=1,BL=4
        MOV AH,0FFH
        MOV SI,0
        MOV DX,380H              ;设置第一片芯片的接口地址
NEXT:   MOV CX,8192
GOON:   IN AL,DX
        TEST AL,BH
```

```
JZ GOON
MOV [SI],AH
INC SI
LOOP GOON
INC DX
SHL BH,1
DEC BL
JNZ NEXT
HLT
```

由例 5-7～例 5-10 可以看出，半导体存储器的设计主要是译码电路的设计。在利用已有存储器芯片构造内存储器时，可以采用多种连接方式。首先通过查阅相关技术手册，了解已有存储器芯片的外部引线含义；在此基础上，根据 CPU 总线所能提供的信号，选择适当的器件构造译码器，就可以很容易设计出任何所需的存储器空间。

习　　题

5.1　什么是存储器系统？微机中的存储器系统主要分为哪几类？它们的设计目标是什么？

5.2　内部存储器主要分为哪两类？它们的主要区别是什么？

5.3　为什么动态 RAM 需要定时刷新？

5.4　CPU 寻址内存的能力最基本的因素取决于________。

5.5　设构成一个存储器系统的两个存储器是 M_1 和 M_2，其存储容量分别为 S_1 和 S_2，访问速度为 T_1、T_2，每 KB 的价格为 C_1、C_2。试问，在什么条件下，该存储器系统的每千字节的价格会接近于 C_2？

5.6　利用全地址译码将 6264 芯片接到 8088 系统总线上，使其所占地址范围为 32000H～33FFFH。

5.7　内存地址从 20000H～8BFFFH 共有多少字节？

5.8　若采用 6264 芯片构成题 5.7 中的内存空间，需要多少片 6264？

5.9　设某微型机内存 RAM 区的容量为 128KB，若用 2164 芯片构成这样的存储器，需多少片 2164？至少需多少根地址线？其中多少根用于片内寻址？多少根用于片选译码？

5.10　现有两片 6116 芯片，所占地址范围为 61000H～61FFFH，试将它们连接到 8088 系统中，并编写测试程序，向所有单元输入一个数据，然后再读出与之比较，若出错则显示"Wrong!"，若全部正确则显示"OK!"。

5.11　什么是字扩展？什么是位扩展？用户自己购买内存条进行内存扩充，是在进行何种存储器扩展？

5.12　74LS138 译码器的接线如图 5-47 所示，试判断其输出端 Y_0、Y_3、Y_5 和 Y_7 所决定

的内存地址范围。

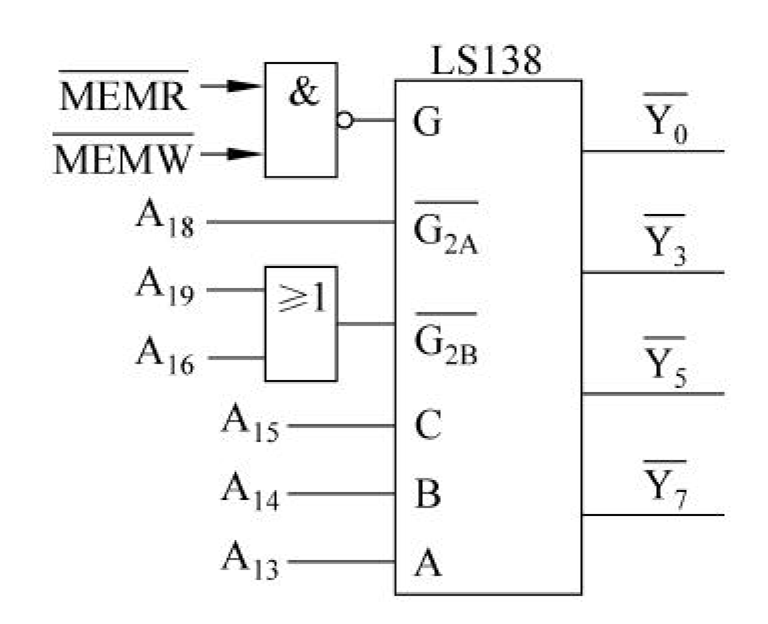


图 5-47　题 5.12 译码器连接图

5.13　某 8088 系统用 2764 ROM 芯片和 6264 SRAM 芯片构成 16KB 的内存。其中，ROM 的地址范围为 FE000H～FFFFFH，RAM 的地址范围为 F0000H～F1FFFH。试利用 74LS138 译码，画出存储器与 CPU 的连接图，并标出总线信号名称。

5.14　叙述 EPROM 的编程过程，并说明 EPROM 和 EEPROM 的不同点。

5.15　试说明 FLASH EEPROM 芯片的特点及 28F040 的编程过程。

5.16　什么是 Cache？它能够极大地提高计算机的处理能力是基于什么原理？

5.17　若主存 DRAM 的存取周期为 70ns，Cache 的存取周期为 5ns，Cache 的命中率为 90%，由它们构成的存储器系统的平均存取周期是多少？

5.18　如何解决 Cache 与主存内容的一致性问题？

5.19　在二级 Cache 系统中，L1 Cache 的主要作用是什么？L2 Cache 呢？

5.20　新购买的或擦除干净的 EPROM 芯片，其各单元的内容是什么？

 章　输入输出和中断技术

引言：

家庭安全防盗系统需要读取监测设备发出的信息（正常或异常），又需要输出相应的报警控制信息。无论是监测设备还是报警器，都属于外部设备。输入输出是计算机与外部设备进行信息交换不可缺少的功能，在整个计算机系统中占有极其重要的地位。计算机所处理的各种信息（包括程序和数据）都要由输入设备提供，而处理的结果则要通过输出设备输出供人们查看。例如键盘、鼠标、扫描仪等都是输入设备，显示器、打印机、绘图仪等都是输出设备。可以说，如果没有输入输出能力，计算机就变得毫无意义。

通过本章的学习，读者应能够在整体上对输入输出系统（Input Output System，I/O系统）、输入输出接口、基本输入输出方法及中断控制技术有一定的了解，并能够利用简单接口芯片实现外设与系统的连接和信息传送。

教学目的：

(1) 了解输入输出系统、输入输出接口和输入输出端口的一般概念；
(2) 了解输入输出端口的编址方式；
(3) 深入理解基本输入输出方法及中断控制技术；
(4) 掌握简单接口芯片的应用。

6.1　输入输出系统概述

计算机在运行过程中所需要的程序和数据都要从外部输入，运算的结果要输出到外部去。在计算机与外部世界进行信息交换的过程中，输入输出系统提供了所需的控制和各种手段。这里的外部世界是指除计算机之外的与计算机交换信息的人和物，如系统操作员、键盘、鼠标、显示器、打印机、辅助存储器等。人们将人以外的各种设备统称为输入输出设备或外围设备。

在计算机系统中，通常把处理器和主存储器之外的部分统称为输入输出系统，它包括输入输出设备、输入输出接口和输入输出软件。

6.1.1　I/O 系统的特点

输入输出系统是计算机系统中最具多样性和复杂性的部分，主要具有以下 4 个方面

的特点。

1. 复杂性

现代计算机输入输出系统的复杂性主要表现在两个方面，一是输入输出设备的复杂性。I/O 设备的品种繁多、功能各异，在工作时序、信号类型、电平形式等各方面都不相同，另外，I/O 设备还涉及机、光、电、磁、自动控制等多种学科。设备的复杂性使得输入输出系统成为计算机系统中最具多样性和复杂性的部分。为了使一般用户能够只通过一些简单命令和程序就能调用和管理各种 I/O 设备，而无须了解设备的具体工作细节，现代计算机系统中都将输入输出系统的复杂性隐藏在操作系统中。

除输入输出设备的复杂性之外，输入输出系统的复杂性还表现在处理器本身和操作系统所产生的一系列随机事件也要调用输入输出系统进行处理，如中断等。

2. 异步性

CPU 的各种操作都是在统一的时钟信号作用下完成的，各种操作都有自己的总线周期，而不同的外部设备也有各自不同的定时与控制逻辑，且大都与 CPU 时序不一致，它们与 CPU 的工作通常都是异步进行的。当某个输入设备有准备好的数据需要向 CPU 传送或输出设备的数据寄存器空可以接收数据时，一般要先向 CPU 提出服务请求，如果 CPU 响应请求，就转去执行相应的服务。对 CPU 来讲，这种请求可能是随机的，每两次这样的请求之间可能间隔很短，也可能相隔时间较长，而且在响应请求之前，外设可能已为“准备好”运行了相当一段时间。如此，输入输出系统相对于 CPU 就存在操作上的异步性和时间上的任意性。

3. 实时性

用作实时控制系统的计算机对时间的要求很高。实时性是指处理器对每一个连接到它的外设或处理器本身，在需要或出现异常时，如电源故障、运算溢出、非法指令等，都要能够给予及时的处理，以防止错过服务时机使数据丢失或产生错误。外部设备的种类很多，信息的传送速率相差也很大，如有的是单字符传送，即每次只传送一个字符，像打印机等，传送速度为每秒几个到几十个字符；而有的则是按数据块或按文件传送，像磁盘等，每秒传送几到几十兆字符。因此，要求输入输出系统能够保证处理器对不同设备提出的请求都能提供及时的服务，这就是输入输出系统的实时性要求。

4. 与设备无关性

由于输入输出设备在信号电平、信号形式、信息格式及时序等方面的差异，使得它们与 CPU 之间不能够直接地连接，而必须通过一个中间环节，这就是输入输出接口（Input Output Interface）。为了适应与不同外设的连接，人们规定了一些独立于具体设备的标准接口，如串行接口、并行接口等。不同型号的外设可根据自己的特点和要求选择一种标准接口与处理器相连。对连接到同一种接口上的外设，它们之间的差异由设备本身的控制器通过软件和硬件来填补。这样，CPU 能够通过统一的软件和硬件来管理各

种各样的外部设备,而不需要了解各种外设的具体细节。例如,在 Windows 9x 操作系统中,凡经过 Microsoft 公司测试过的机型和外设都可直接相连,由操作系统统一进行管理。

6.1.2 I/O 接口的基本功能

微型计算机上的所有部件都是通过总线互联的,外部设备也不例外。I/O 接口就是将外设连接到系统总线上的一组逻辑电路的总称,也称为外设接口。在一个实际的计算机控制系统中,CPU 与外部设备之间常需要进行频繁的信息交换,包括数据的输入输出、外部设备状态信息的读取及控制命令的传送等,这些都是通过接口来实现的。

1. I/O 接口要解决的问题

外部设备的种类繁多,有机械式、电动式、电子式和其他形式。它们涉及的信息类型也不相同,可以是数字量、模拟量或开关量。因此 CPU 与外设之间交换信息时需要解决以下问题。

(1) 速度匹配问题。CPU 的速度很高,而外设的速度有高有低,而且不同的外设速度差异甚大。

(2) 信号电平和驱动能力问题。CPU 的信号都是 TTL 电平(一般在 0～5V 之间),而且提供的功率很小,而外设需要的电平要比这个范围宽得多,需要的驱动功率也较大。

(3) 信号形式匹配问题。CPU 只能处理数字信号,而外设的信号形式多种多样,有数字量、开关量、模拟量(电流、电压、频率、相位),甚至还有非电量,如压力、流量、温度、速度等。

(4) 信息格式问题。CPU 在系统总线传送的是 8 位、16 位或 32 位并行二进制数据,而外设使用的信号形式信息格式各不相同。有些外设是数字量或开关量,而有些外设使用的是模拟量;有些外设采用电流量,而有些是电压量;有些外设采用并行数据,而有些则是串行数据。

(5) 时序匹配问题。CPU 的各种操作都是在统一的时钟信号作用下完成的,各种操作都有自己的总线周期,而各种外设也有自己的定时与控制逻辑,大都与 CPU 时序不一致。因此各种各样的外设不能直接与 CPU 的系统总线相连。

在计算机中,上述问题是通过在 CPU 与外设之间设置相应的 I/O 接口电路来予以解决的。

2. I/O 接口的功能

由 I/O 接口在系统中的位置可以得出接口电路应具有如下功能。

(1) I/O 地址译码与设备选择。所有外设都通过 I/O 接口挂接在系统总线上,在同一时刻,总线只允许一个外设与 CPU 进行数据传送。因此,只有通过地址译码选中的 I/O 接口允许与总线相通,而未被选中的 I/O 接口呈现为高阻状态,与总线隔离。

(2) 信息的输入输出。通过 I/O 接口,CPU 可以从外部设备输入各种信息,也可将

处理结果输出到外设;CPU 可以控制 I/O 接口的工作(向 I/O 接口写入命令),还可以随时监测与管理 I/O 接口和外设的工作状态;必要时,I/O 口还可以通过接口向 CPU 发出中断请求。

(3) 命令、数据和状态的缓冲与锁存。因为 CPU 与外设之间的时序和速度差异很大,为了能够确保计算机和外设之间可靠地进行信息传送,要求接口电路应具有信息缓冲能力。接口不仅应缓存 CPU 送给外设的信息,也要缓存外设送给 CPU 的信息,以实现 CPU 与外设之间信息交换的同步。

(4) 信息转换。I/O 接口还要实现信息格式变换、电平转换、码制转换、传送管理以及联络控制等功能。

6.1.3 I/O 端口的编址方式

CPU 与 I/O 接口进行通信实际上是通过 I/O 接口内部的一组寄存器①实现的,这些寄存器通常称为 I/O 端口(I/O Port)。I/O 端口包括 3 种类型: 数据端口、状态端口和命令(或控制)端口,根据需要,一个 I/O 接口可能仅包含其中的一类或两类端口,当然也可能包含全部三类端口。CPU 通过数据端口从外设读入数据(或向外设输出数据),从状态端口读入设备的当前状态,通过命令(控制)端口向外设发出控制命令。

8088/8086 CPU 最多能够管理 64K 个端口(只使用地址总线的 $A_0 \sim A_{15}$),那么当前的操作是针对哪一个端口呢? 要确定这一点,就要像为内存单元分配地址那样为每个端口分配一个地址(称为 I/O 地址)。因为一个外设总是对应着一个或多个端口,所以有时也将端口地址称为外设地址。当一个外设有多个端口时,为管理方便,通常是为其分配一个连续的地址块,这个地址块中最小的那个地址称为(外设的)基地址(Base Address)。

在微型计算机系统中,I/O 端口的编址通常有两种不同的方式: 与内存单元统一编址、独立编址。

1. I/O 端口与内存单元统一编址

I/O 端口与内存单元统一编址方式又称为存储器映射编址方式,即把每个 I/O 端口都当做一个存储单元看待,端口与存储器单元在同一个地址空间中进行编址。通常是在整个地址空间中划分出一小块连续的地址分配给 I/O 端口。被端口占用了的地址,存储器不能再使用。图 6-1 给出了 I/O 端口与内存单元统一编址的示意图。图中,分配给 I/O 端口的地址范围为 F0000H~FFFFFH,共 65536 个地址。

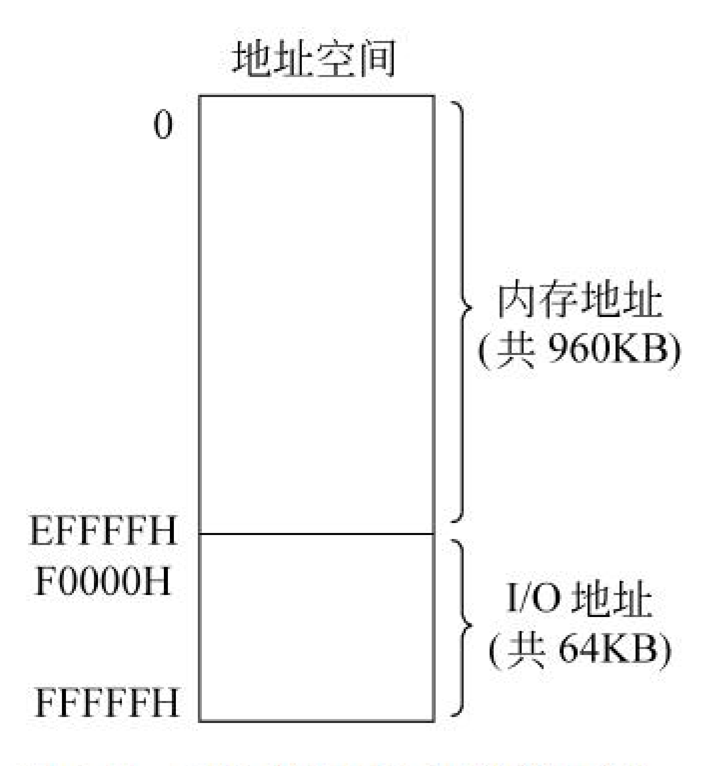


图 6-1 I/O 端口与内存单元统一编址示意图

统一编址方式的优点是可以用访问内存的方法来

① 简单的接口也可仅由三态门构成,但要求传输过程未完成之前信号应保持不变。

访问I/O端口。由于访问内存的指令种类丰富、寻址方式多样,因此这种编址方式为访问外设带来了很大的灵活性。从理论上讲,所有用于内存的指令都可以用于外设,不再需要专门的I/O指令。同时,I/O控制信号也可与存储器的控制信号共用,从而给应用带来了很大的方便。

统一编址方式的缺点是外设占用了一部分内存地址空间,这就减少了内存可用的地址范围,因此对内存容量有潜在的影响。此外从指令上不易区分当前是对内存进行操作还是对外设进行操作。

2. I/O端口独立编址

I/O端口独立编址时,内存地址空间和外设地址空间是相互独立的。例如,8086/8088系统的内存地址范围为00000H~FFFFFH,而外设端口的地址范围为0000H~FFFFH,这两个地址空间相互独立,互不影响。CPU在寻址内存和外设时,使用不同的控制信号来区分当前是对内存操作还是对I/O端口操作。从第2章中8088 CPU引脚功能部分已知,当8088的$IO/\overline{M}$信号为低电平时,表示当前CPU执行的是存储器读写操作,这时地址总线上给出的是某个存储单元的地址;当$IO/\overline{M}$信号为高电平时,则表示当前CPU执行的是I/O读写操作,这时地址总线上给出的是某个I/O端口的地址。另外,采用I/O端口独立编址的CPU,其指令系统中单独设置有专用的I/O指令,用于对I/O端口进行读写操作。但这些指令的功能比较弱,一些操作必须将数据由外设首先读入到CPU的寄存(累加)器后才能进行。

综上所述,I/O端口独立编址的特点如下。

(1) I/O端口的地址空间与内存地址空间完全独立。

(2) I/O端口与内存使用不同的控制信号。

(3) 指令系统中设置了专门用于访问外设的I/O指令。

I/O端口独立编址方式在Z80系列及Intel公司的x86系列CPU中得到广泛采用。8086/8088 CPU就采用了I/O端口独立编址方式。

6.1.4 I/O端口地址的译码

在IBM PC中,所有输入输出接口与CPU之间的通信都是由I/O指令来完成的。在执行I/O指令时,CPU首先需要将要访问端口的地址放到地址总线上(即选中该端口),然后才能对其进行读写操作。将总线上的地址信号转换为某个端口的“使能”(Enable)信号,这个操作就称为端口地址的译码。

有关译码的技术在第5章已经接触过。对第5章中讨论的存储器系统,使一个存储器芯片在整个存储空间中占据一定的地址范围是通过高位地址信号的译码来确定的。那么,在输入输出技术中,端口的地址也是通过地址信号的译码来确定的。只是有以下几点要注意。

(1) 8088 CPU能够寻址的内存空间为1MB,故地址总线的全部20根信号线都要使用,其中高位(A_{19}~A_i)用于确定芯片的地址范围,而低位(A_{i-1}~A_0)用于片内寻址;而

8088 CPU 能够寻址的 I/O 端口仅为 64K(65535)个，故只使用了地址总线的低 16 位信号线。对只有单一 I/O 地址(端口)的外设，这 16 位地址线一般应全部参与译码，译码输出直接选择该外设的端口；对具有多个 I/O 地址(端口)的外设，则 16 位地址线的高位参与译码(决定外设的基地址)，而低位则用于确定要访问哪一个端口。

(2) 当 CPU 工作在最大模式时，对存储器的读写要求控制信号 $\overline{\text{MEMR}}$ 或 $\overline{\text{MEMW}}$ 有效；如果是对 I/O 端口读写，则要求控制信号 $\overline{\text{IOR}}$ 或 $\overline{\text{IOW}}$ 有效。

(3) 地址总线上呈现的信号是内存的地址还是 I/O 端口的地址取决于 8088 CPU 的 IO/$\overline{\text{M}}$ 引脚的状态。当 IO/$\overline{\text{M}}$=0 时为内存地址，即 CPU 正在对内存进行读写操作；IO/$\overline{\text{M}}$=1 为 I/O 端口地址，即 CPU 正在对 I/O 端口进行读写操作。

I/O 地址译码的方式是多种多样的，综合起来主要可分为两种：用基本逻辑门电路构成译码器或用专门的译码器进行译码。译码电路与存储器的译码电路基本相同，这些在第 5 章已经介绍，此处不再赘述。

6.2 简单接口电路

6.2.1 接口电路的基本构成

CPU 通过接口与外部设备的连接示意图如图 6-2 所示。通过接口传送的信息除数据外，还有反映当前外设工作状态的状态信息以及 CPU 向外设发出的各种控制信息。

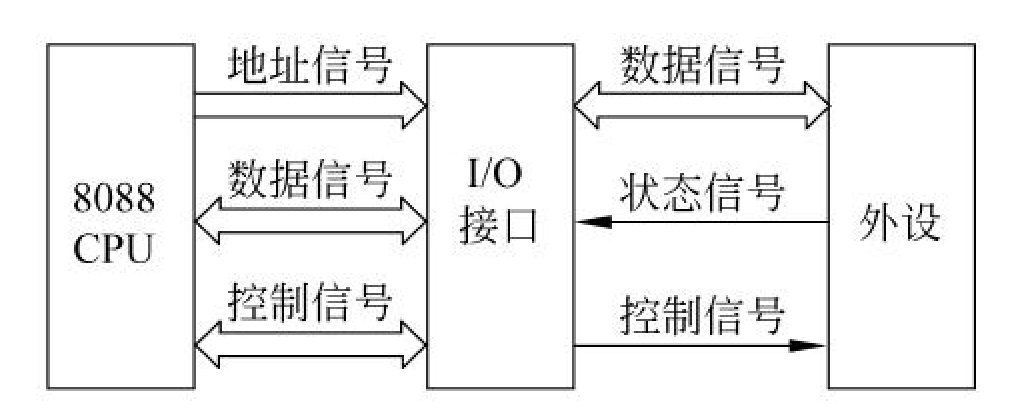


图 6-2 CPU 与外设之间的接口

负责把信息从外部设备送入 CPU 的接口(端口)叫做输入接口(端口)，而将信息从 CPU 输出到外部设备的接口(端口)则称为输出接口(端口)。

在输入数据时，由于外设处理数据的时间一般要比 CPU 长得多，数据在外部总线上保持的时间相对较长，所以要求输入接口必须要具有对数据的控制能力，即只有当外部数据准备好、CPU 可以读取时才将数据送上系统数据总线。若外设本身具有数据保持能力，通常可以仅用一个三态门缓冲器作为输入接口。当三态门控制端信号有效时，三态门导通，外设与数据总线连通，CPU 将外设准备好的数据读入；当其控制端信号无效时，三态门断开，该外设就从数据总线脱离，数据总线又可用于其他信息的传送。

在输出数据时，同样由于外设的速度比较慢，要使数据能正确写入外设，CPU 输出的数据一定要能够保持一段时间。如果这个“保持”的工作由 CPU 来完成，则对其资源就

必然是个浪费。实际上，从前面介绍的“总线写”时序图可以看出，CPU 送到总线上的数据只能保持几微秒。因此，要求输出接口必须要具有数据的锁存能力。CPU 输出的数据通过总线送入接口锁存，由接口将数据一直保持到被外设取走。简单的输出接口一般由锁存器构成。

以上三态门和锁存器的控制端一般与 I/O 地址译码器的输出信号线相连，当 CPU 执行 I/O 指令时，指令中指定的 I/O 地址经译码后使控制信号有效，打开三态门（对外设读时）或触发锁存器导通，将数据锁入锁存器（对外设写时）。

本节将介绍一些结构简单又较常用的通用接口芯片，并通过举例说明它们的使用方法。

6.2.2 三态门接口

一个典型的三态门芯片 74LS244 如图 6-3 所示。从图中不难看出该芯片由 8 个三态门构成。74LS244 有两个控制端：E_1 和 E_2。每个控制端各控制 4 个三态门。当某一控制端有效（低电平）时，相应的 4 个三态门导通；否则，相应的三态门呈现高阻状态（断开）。实际使用中，通常是将两个控制端并联，这样就可用一个控制信号来使 8 个三态门同时导通或同时断开[①]。

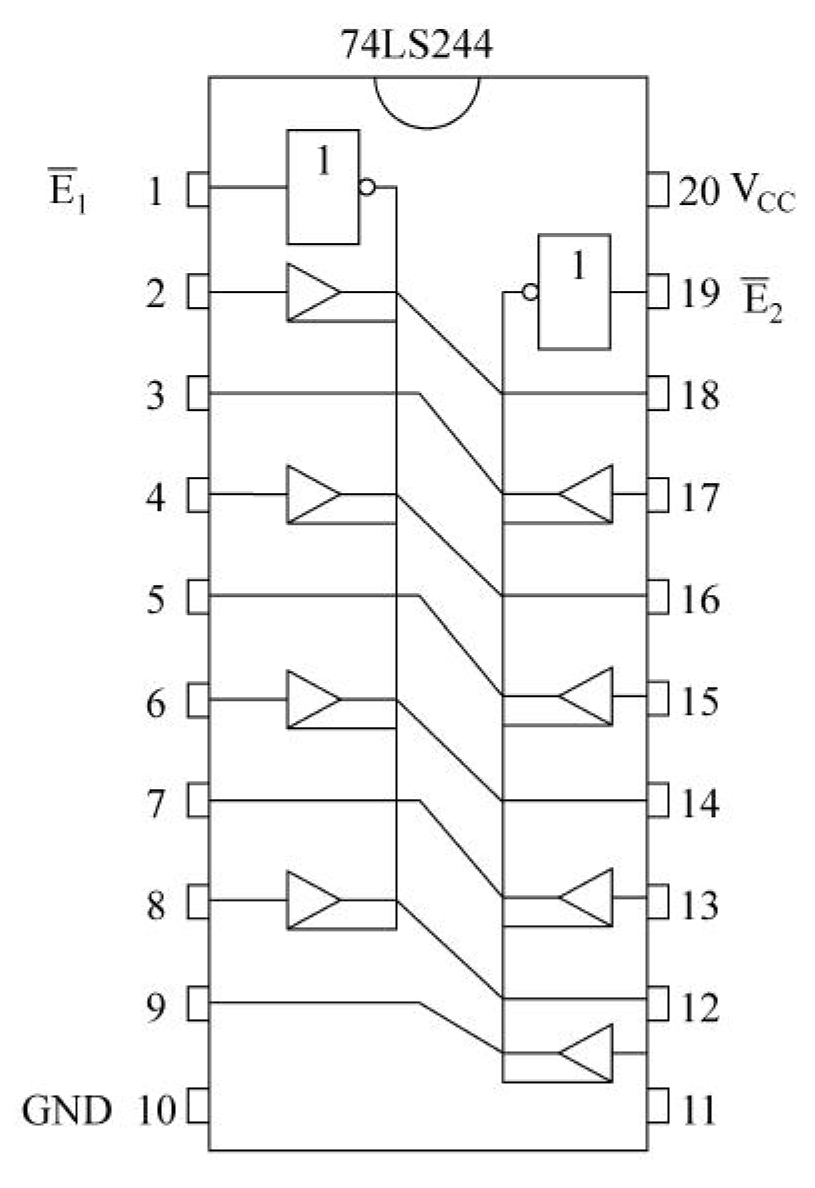


图 6-3　74LS244 芯片引脚图

由于三态门具有“通断”控制能力的这个特点，故可利用其作输入接口。利用三态门作为输入信号接口时，要求信号的状态是能够保持的，这是因为三态门本身没有对信号的保持或锁存能力。图 6-4 是一个利用三态门 74LS244 作为开关量输入接口的例子。图中，74LS244 的输入端接有 8 个开关 K_0、K_1、…、K_7。当 CPU 读该接口时，总线上的 16 位地址信号通过译码使 E_1 和 E_2 有效，三态门导通，8 个开关的状态经数据线 $D_0 \sim D_7$ 被读入到 CPU 中。这样，就可测量出这些开关当前的状态是打开还是闭合。当 CPU 不读此接口地址时，E_1 和 E_2 为高电平，则三态门的输出为高阻状态，使其与数据总线断开。

用一片 74LS244 芯片作为输入接口最多可以连接 8 个开关或其他具有信号保持能力的外设。当然也可只接一个外设而让其他端悬空，对空着未用的端，其对应位的数据是任意值，在程序中常用逻辑“与”指令将其屏蔽掉。

如果有更多的开关状态（或其他外设）需要输入时，可用类似的方法用两片或更多的芯片并联使用。

① 大多数 I/O 操作每次同时传送 8 位数据。

74LS244 芯片除用作输入接口外，还常用来作为信号的驱动器。

【例 6-1】 编写程序判断图 6-4 中的开关的状态。如果所有的开关都闭合，则程序转向标号为 NEXT1 的程序段执行，否则转向标号为 NEXT2 的程序段执行。

图 6-4 中，作为输入接口的三态门 74LS244，其 I/O 地址采用了部分地址译码，地址线 A_1 和 A_0 未参加译码，故它所占用的地址为 83FCH～83FFH。可以用其中任何一个地址，而其他重叠的 3 个地址空着不用。另外，由图可以看出，当开关闭合时输入低电平（=0）。判断所有开关是否全部闭合的程序段如下：

```
MOV     DX,83FCH
IN      AL,DX
AND     AL,0FFH
JZ      NEXT1
……
```

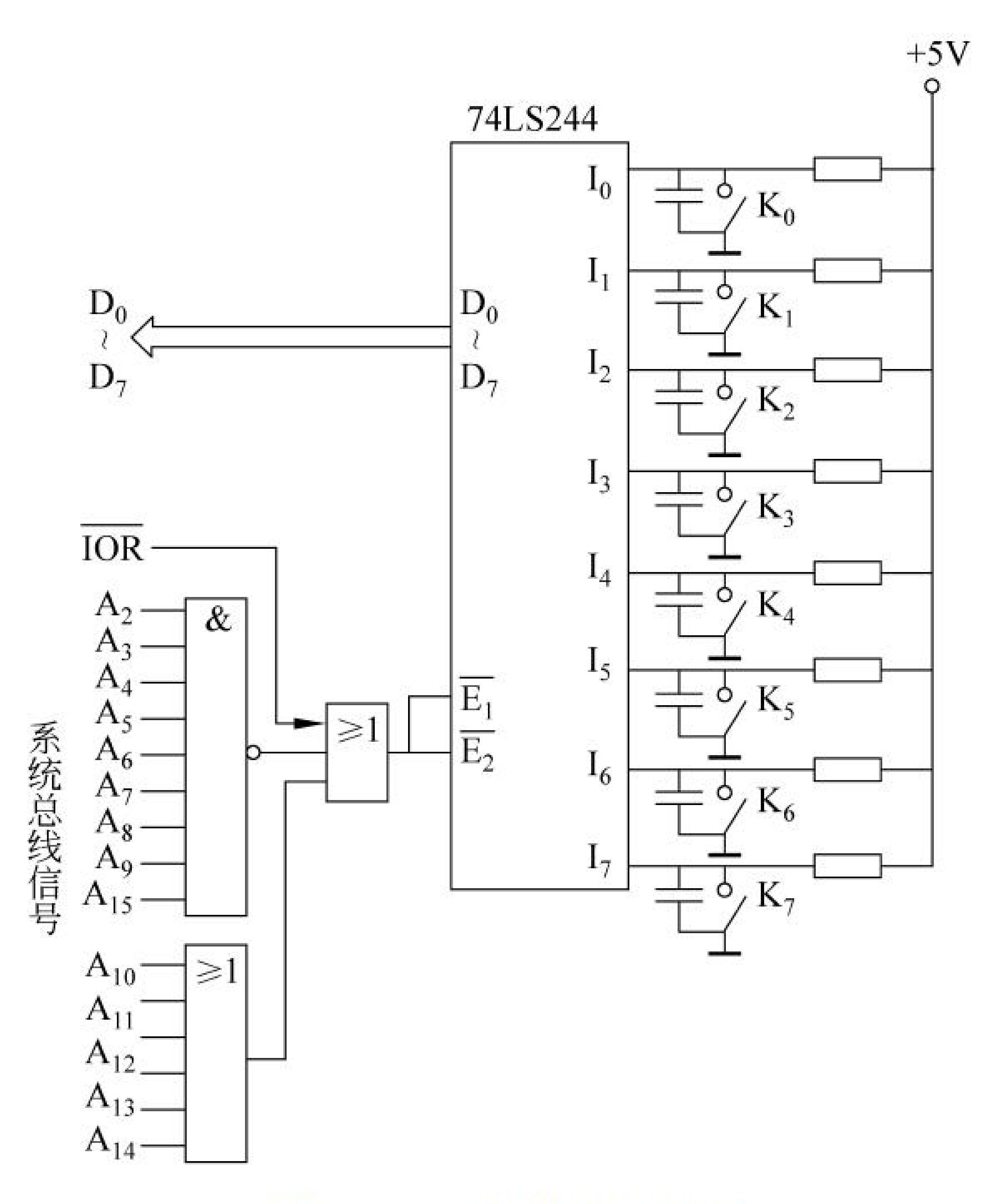
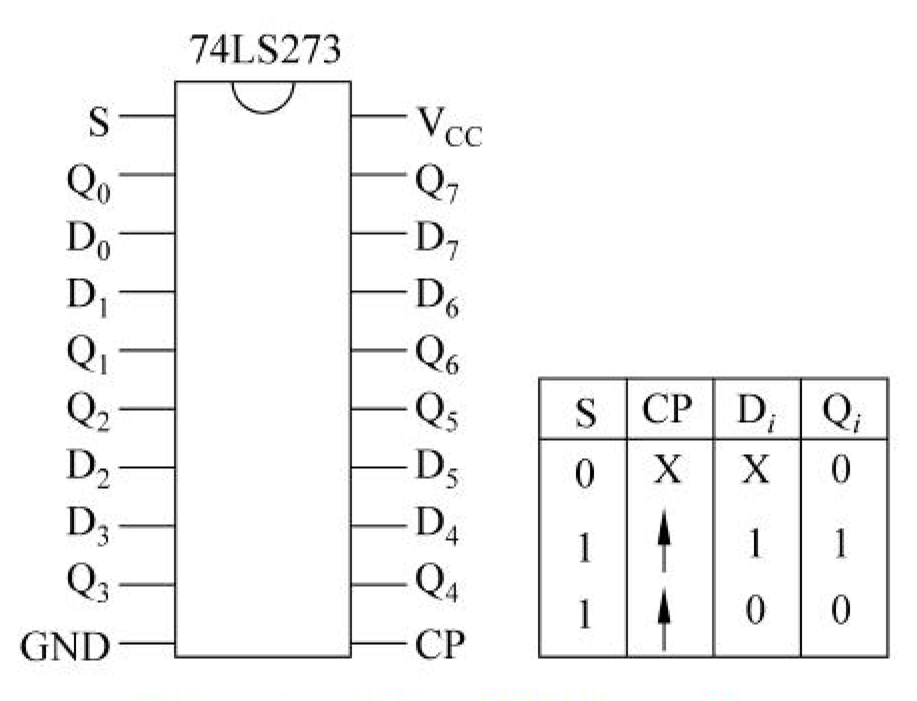


图 6-4　三态门作输入接口

6.2.3 锁存器接口

由于三态门器件不具备数据的保存（或称锁存）能力，要求信号源能够将信号保持足够长的时间直到被 CPU 读取，所以它一般只用作输入接口，而不能直接用作数据输出接口。

数据输出接口通常采用具有信息存储能力的双稳态触发器来实现。最简单的输出接口可用 D 触发器构成。例如，常用的锁存器 74LS273，它内部包含了 8 个 D 触发器，其引线图及真值表如图 6-5 所示。74LS273 共有 8 个数据输入端（$D_0 \sim D_7$）和 8 个数据输出端 $Q_0 \sim Q_7$。S 为复位端，低电平有效。CP 为脉冲输入端，在每个脉冲的上升沿将输入端 D_i 的状态锁存在 Q_i 输出端，并将此状态保持到下一个时钟脉冲的上升沿。74LS273 常用来作为并行输出接口。另外，使用其中的某一个 D 触发器也可通过软件编程实现简单的串行输出。



S	CP	D_i	Q_i
0	X	X	0
1	↑	1	1
1	↑	0	0

图 6-5　74LS273 引线图和真值表

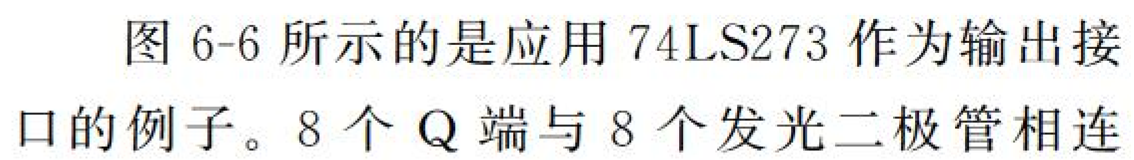


图 6-6 所示的是应用 74LS273 作为输出接口的例子。8 个 Q 端与 8 个发光二极管相连接，假设要使接到 Q_0 端和 Q_6 端的发光二极管发光，其对应的 Q_0、Q_6 端须输出“1”状态，而其他 Q 端则输出“0”状态。假定该输出接口的地址为 0FFFFH，则程序段如下：

```
MOV DX,0FFFFH
MOV AL,01000001B
OUT DX,AL
```

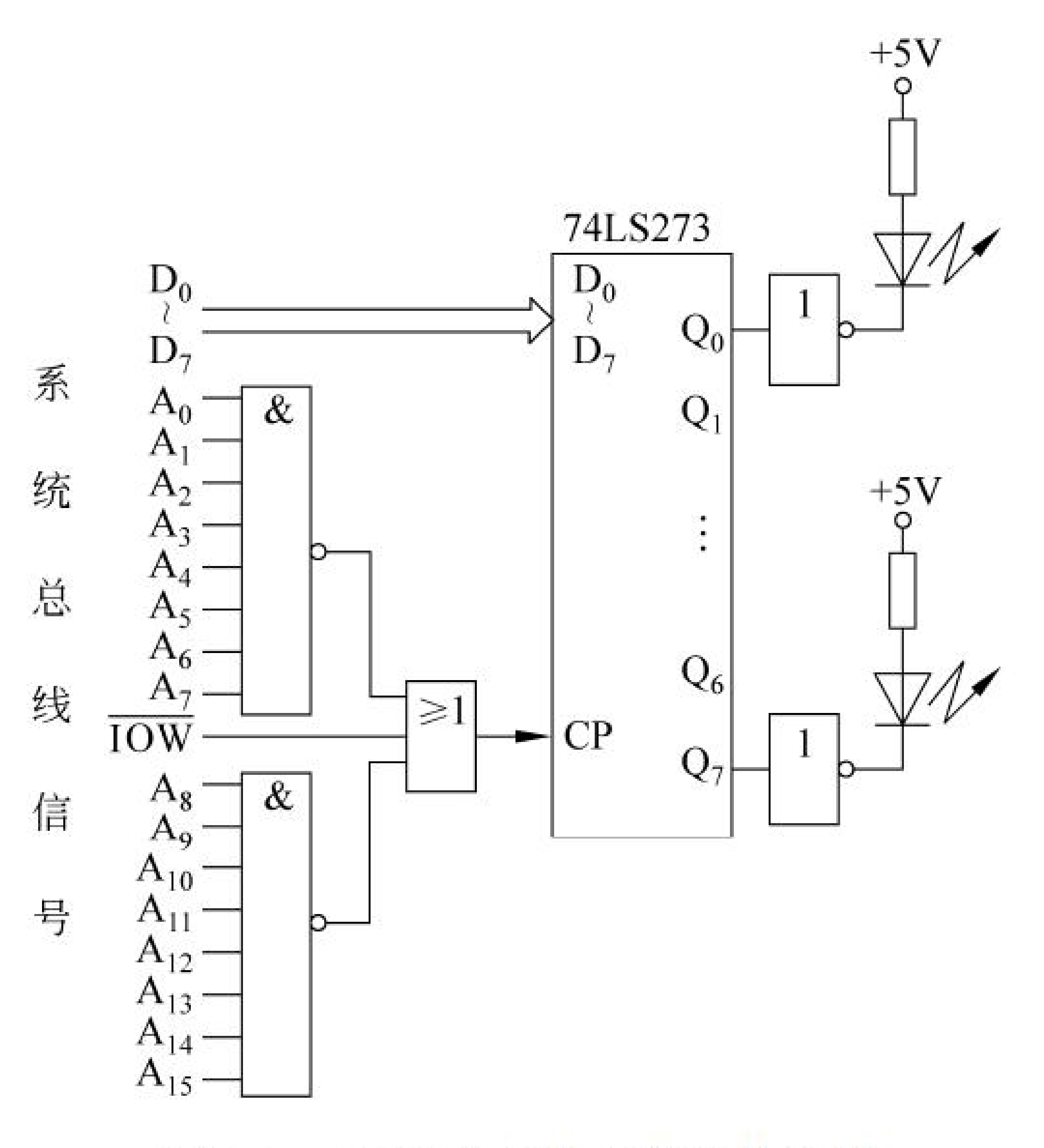
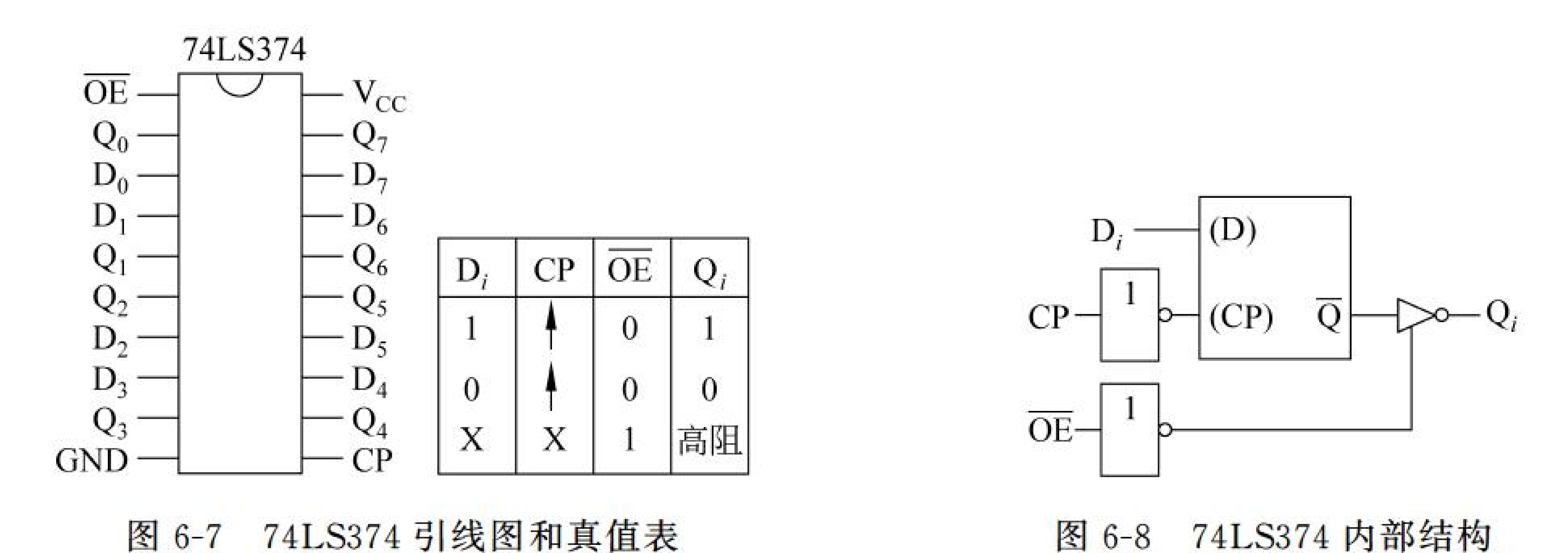


图 6-6　74LS273 作输出接口的应用

74LS273 的数据锁存输出端 Q 是通过一个一般的门（二态门）输出的。也就是说，只要 74LS273 正常工作，其 Q 端总有一个确定的逻辑状态（0 或 1）输出。因此，74LS273 无法直接用作输入接口，即它的 Q_i 端绝对不允许直接与系统的数据总线相连接。那么，有没有既可用作输入接口又能用作输出接口的芯片呢？回答是肯定的。下面介绍一种带有

三态输出的锁存器 74LS374，这也是经常用到的一种电路芯片，其引线图和真值表如图 6-7 所示。从引线上可以看出，它比 74LS273 多了一个输出允许端$\overline{OE}$。只有当$\overline{OE}=0$时，74LS374 的输出三态门才导通；$\overline{OE}=1$ 时，则呈高阻状态。图 6-8 所示为 74LS374 中一个锁存器的内部结构图，由图可知，74LS374 在 D 触发器输出端加有一个三态门。



D_i	CP	$\overline{OE}$	Q_i
1	↑	0	1
0	↑	0	0
X	X	1	高阻

图 6-7　74LS374 引线图和真值表　　　　图 6-8　74LS374 内部结构

74LS374 在用作输入接口时，端口地址信号经译码电路接到$\overline{OE}$端，外设数据由外设提供的选通脉冲锁存在 74LS374 内部。当 CPU 读该接口时，译码器输出低电平，使 74LS374 的输出三态门打开，读出外设的数据。如果 74LS374 用作输出接口，也可将$\overline{OE}$端接地，使其输出三态门一直处于导通状态，这样就与 74LS273 一样使用了。

分别用 74LS374 作为输入和输出接口的电路如图 6-9 所示。

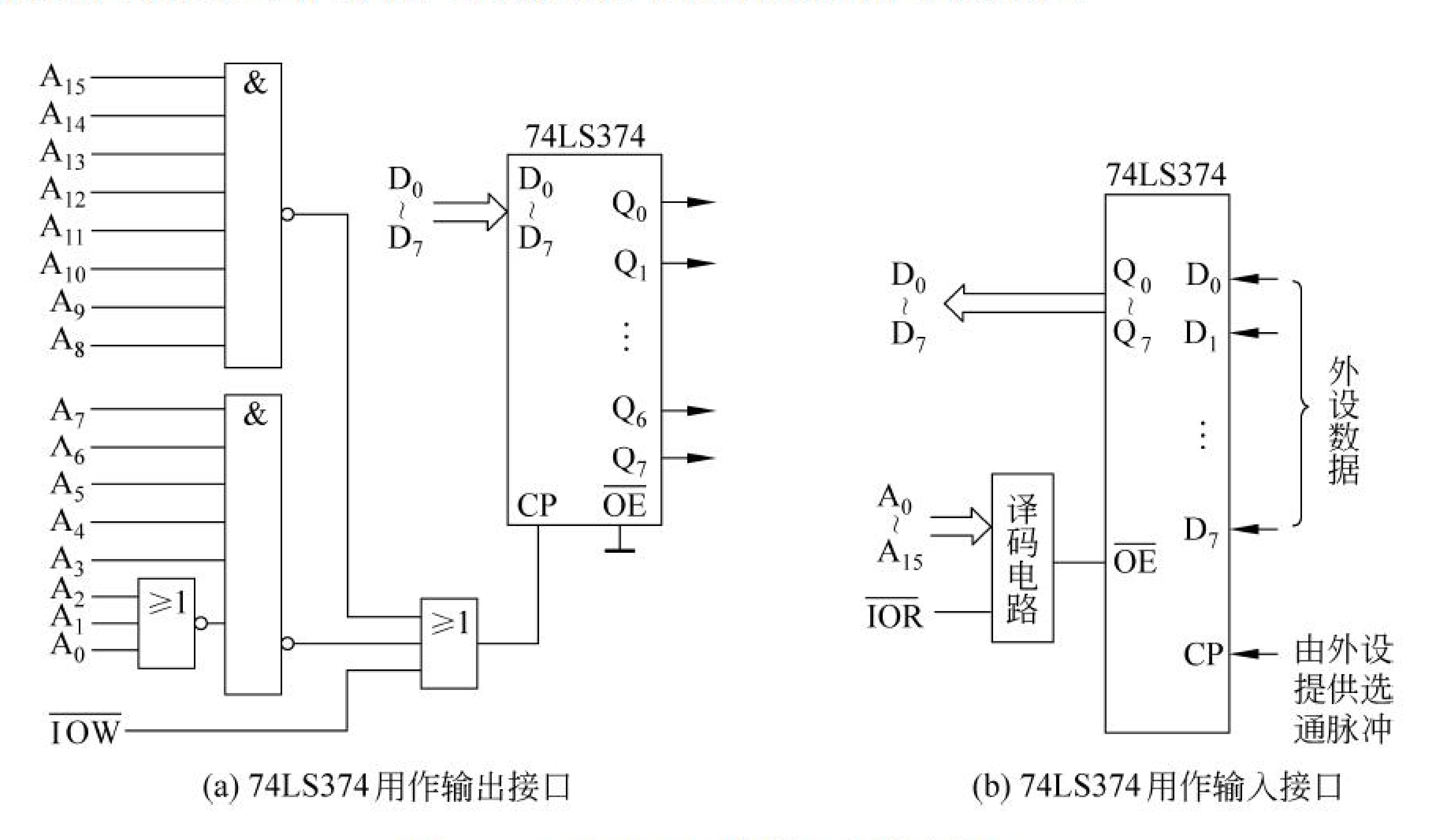


图 6-9　74LS374 用作输入和输出接口

另外还有一种常用的带有三态门的锁存器芯片 74LS373，它与 74LS374 在结构和功能上完全一样，区别是数据锁存的时机不同，带有三态门的锁存器芯片 74LS373 是在 CP 脉冲的高电平期间将数据锁存。

总之，简单接口电路芯片在构造上比较简单，使用也很方便，常作为一些功能简单的外部设备的接口电路。但由于它们的功能有限，对较复杂的功能要求就难以胜任。在后

面的几节中，还将介绍一些功能较强的可编程的接口芯片。

6.2.4 简单接口的应用举例

在本小节中，利用 74LS244 和 74LS273 作为输入和输出接口，通过编写程序，控制 LED 数码管显示不同的数字或符号。

1. LED 数码管

LED 数码管分为共阳极和共阴极两种结构，在封装上有将一位、二位或更多位封装在一起的。由于篇幅限制，这里只介绍一种共阳极封装的 LED 数码管，如图 6-10 所示。当某一段的发光二极管流过一定电流（例如 10mA 左右）时，它所对应的段就发光；而无电流流过时，则不发光。不同发光段的组合就可显示出不同的数字和符号。

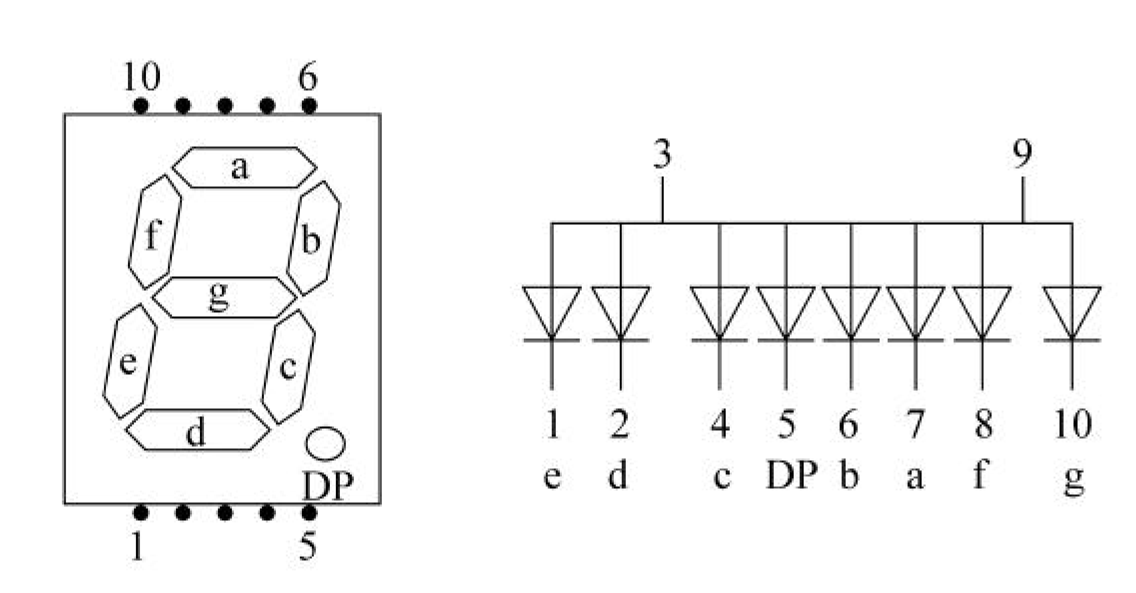


图 6-10 共阳极 LED 数码管示意图

2. 应用与连接

七段 LED 数码管作为一种外设与系统总线有多种接口方式，这里利用前面学到的 74LS273 作为输出接口，用集电极开路门 7406 作为驱动器与 LED 数码管连接。另外，采用 74LS244 作为输入接口，输入开关的状态。线路连接如图 6-11 所示。图中的电路功能是：当开关 K 处于闭合状态时，在 LED 数码管上显示“0”；当开关 K 处于断开状态时，在 LED 数码管上显示“1”。与硬件电路相配合完成此功能的程序段如下：

```
FOREVER: MOV   DX,0F1H        ;输入端口地址为 0F1H
         IN    AL,DX          ;读入开关状态
         TEST  AL,1           ;判断开关状态
         MOV   AL,3FH         ;显示“0”
         JZ    DISP
         MOV   AL,06H         ;显示“1”
DISP:    MOV   DX,0F0H        ;输出端口地址为 0F0H
         OUT   DX,AL
         JMP   FOREVER
```

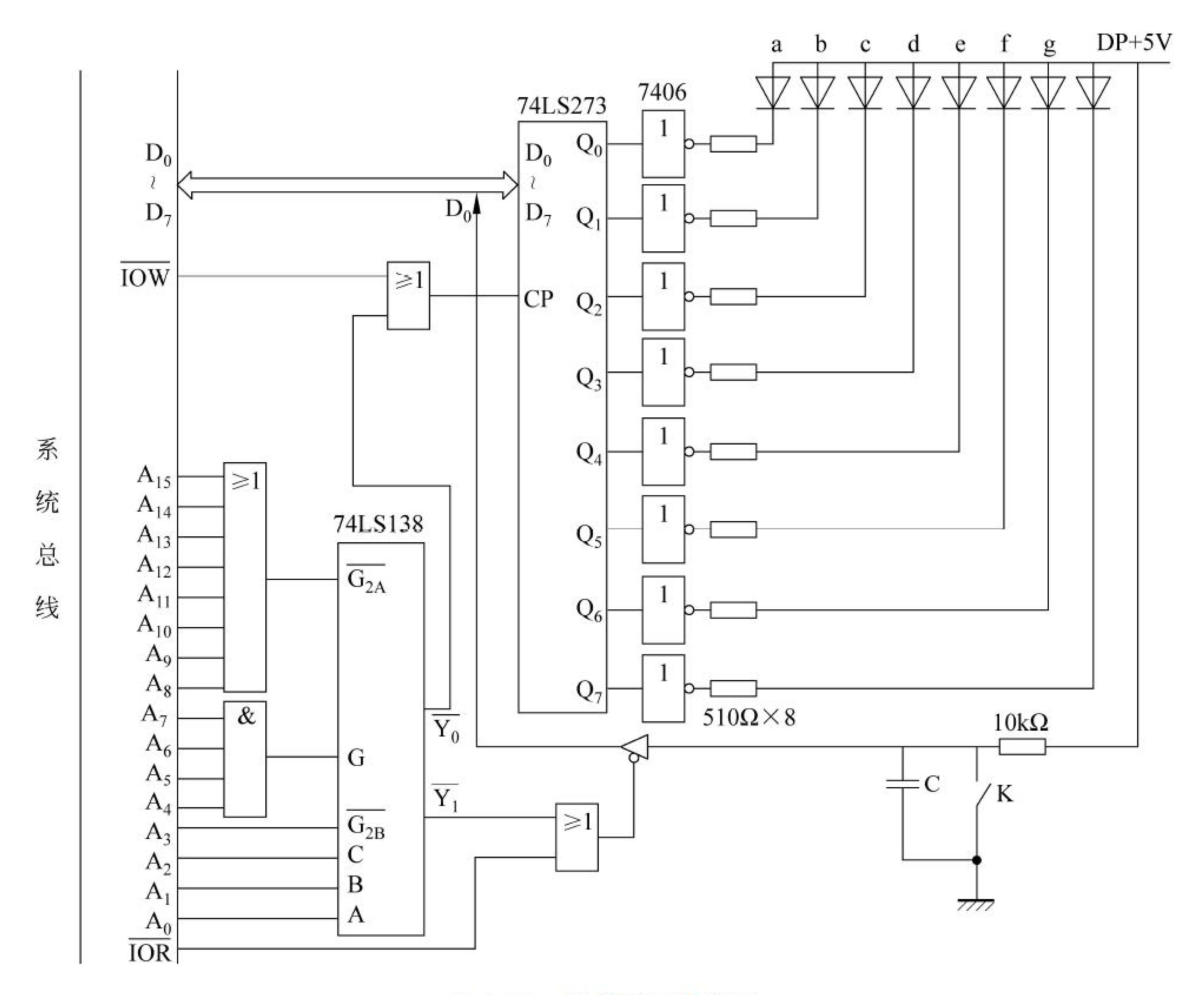


图 6-11　简单接口的应用

6.3　基本输入输出方式

微型计算机系统中，主机与外设之间数据的输入输出方式主要有以下 4 种：无条件传送、查询、中断和直接存储器存取(DMA)方式。

6.3.1　无条件传送方式

无条件传送方式主要用于外部控制过程的各种动作是固定的而且是已知的，控制的对象是一些简单的、随时“准备好”的外设。也就是说在这些设备工作时，随时都可以接收CPU 输出的数据或者它们的数据随时都可以被 CPU 读出，即 CPU 可以不必查询外设当前的状态而无条件地进行数据的输入输出。在与这样的外设交换数据的过程中，数据交换与指令的执行是同步的，因此这种方式也可称为同步传送方式。

当 CPU 从外部设备读入数据时，CPU 执行一条 IN 指令，将低 16 位地址信号组成的端口地址送上地址总线，经过译码，选中对应的端口，然后在$\overline{IOR}=0$ 期间将数据读入

CPU。输出的过程类似,只是必须在$\overline{IOW}$有效时将数据写入外设。下面来看两个无条件数据传送的例子。

图 6-12 中的开关 K 是一个简单的外部设备,它的状态是确定的,要么闭合,要么打开。当计算机通过三态门接口进行读操作时,就读入了开关 K 在指令执行时刻的状态。如果输入数据的 $D_0=0$,就表示 K 处于闭合状态;若 $D_0=1$,则开关 K 处于断开状态。

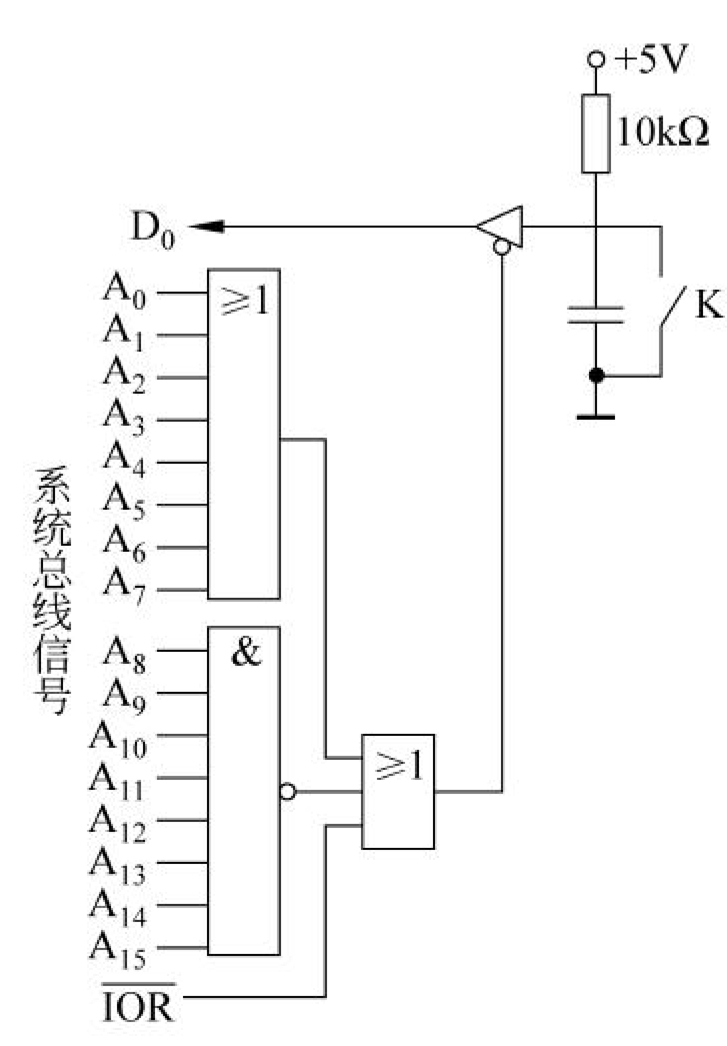


图 6-12 开关 K 通过三态门接口与系统的连接

图 6-6 中的发光二极管也是一个简单外部设备,其状态也是确定的。当锁存器的 Q 端输出高电平时,发光二极管亮;输出低电平时,发光管就不亮,即作为外部设备的发光二极管处于随时可以接收数据的状态。

从以上两个例子可以看出,对于像开关、发光二极管等这一类简单设备来说,它们在某一时刻的状态是固定的,也可以说它们总是准备好的。在读接口时,总可以读到那时开关 K 的状态。写锁存器时,发光二极管总准备好随时接收发来的数据,点亮或熄灭。

对这一类总具有固定状态的简单外部设备的控制,可以采用无条件的传送方式。同类型的设备还有如继电器、步进电机等。

6.3.2 查询方式

对于那些慢速的或总是“准备好”的外设,当它们与 CPU 同步工作时,采用无条件传送方式是适用的,也是很方便的。但在实际应用中,大多数的外设并不是总处于“准备好”状态,在 CPU 需要与它们进行数据交换时,它们或许并不一定满足可进行数据交换的条件,即并不处于“准备好”状态。对这类外设,CPU 在数据传送前必须要先查询一下外设的状态,若准备好才传送数据,否则 CPU 就要等待,直到外设准备好为止。这种利用程序不断地询问外部设备的状态,根据它们所处的状态来实现数据的输入和输出的方式就称为程序查询方式。为了实现这种工作方式,外部设备需向计算机提供一个状态信息,相应的接口除传送数据外,还要有一个传送状态的端口。

图 6-2 所示的其实就是采用查询方式进行数据传送的工作示意图。图中,接口与外设之间有 3 类信息传送,一类是输入或输出的数据,一类是外部设备的状态信息,最后一类是 CPU 通过接口发出的控制信号。工作中,CPU 不断查询外设的状态,判断外设是否准备好进行数据传送。必要时还需送出控制信号。这些将在后面的章节中进一步说明。

图 6-2 的示意图中仅连接了一个外部设备。对这种单一外设采用查询方式进行数据传送的工作过程可描述如下(以 CPU 从外设接收数据为例):①首先查询外设的状态,看数据是否准备好;②若没有准备好,则继续查询;③否则就进行一次数据读取;④数据读

入后,CPU 向外设发出响应信号,表示数据已被接收,外设收到响应信号之后,即开始下一个数据的准备工作;⑤CPU 判断是否已读取完全部数据,若没有读完,则重新进行①,否则就结束传送。

若 CPU 需要向外设输出一个数据,同样首先查询外设的状态,看其是否空闲。若正忙,则等待;若外设准备就绪,处于空闲状态,则 CPU 向外设送出数据和输出就绪信号。就绪信号用来通知外设已送来有效数据。外设接收数据后,向 CPU 发出数据已收到的状态信息。这样,一个数据的输出过程就结束了。

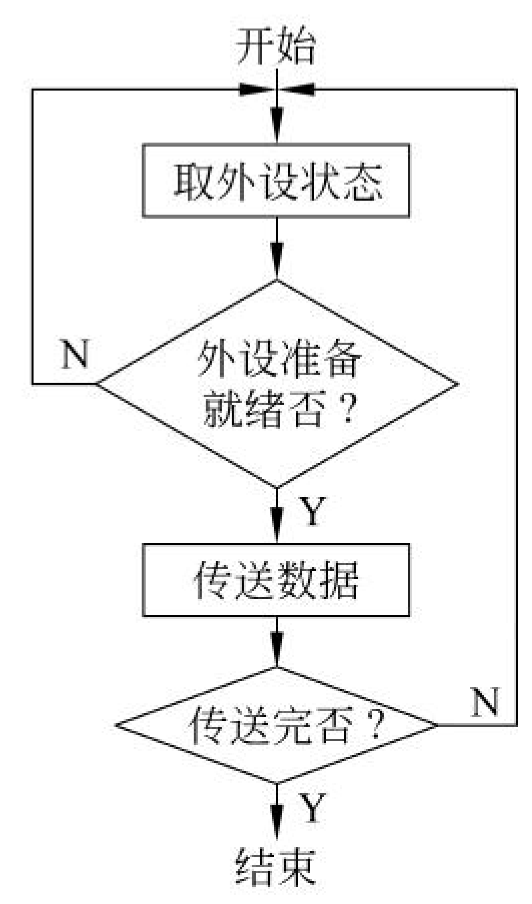


图 6-13 单一外设时的查询方式流程图

上述查询方式的工作流程图如图 6-13 所示。

现在,可以考虑利用上节介绍的三态门接口和锁存器接口,采用查询工作方式,来完成我们的“家庭安全防盗系统”设计了。

“家庭安全防盗系统”设计方案示例 1:

设计基于如下假设:监测装置输出电平信号。当出现异常时,监测装置输出高电平(1),正常状态则输出低电平(0)。

基本方案:利用三态门接口读取 8 个监测装置的输出信息,用锁存器接口控制报警灯闪烁和报警器发声,可以设计如图 6-14 所示的系统连接示意图。

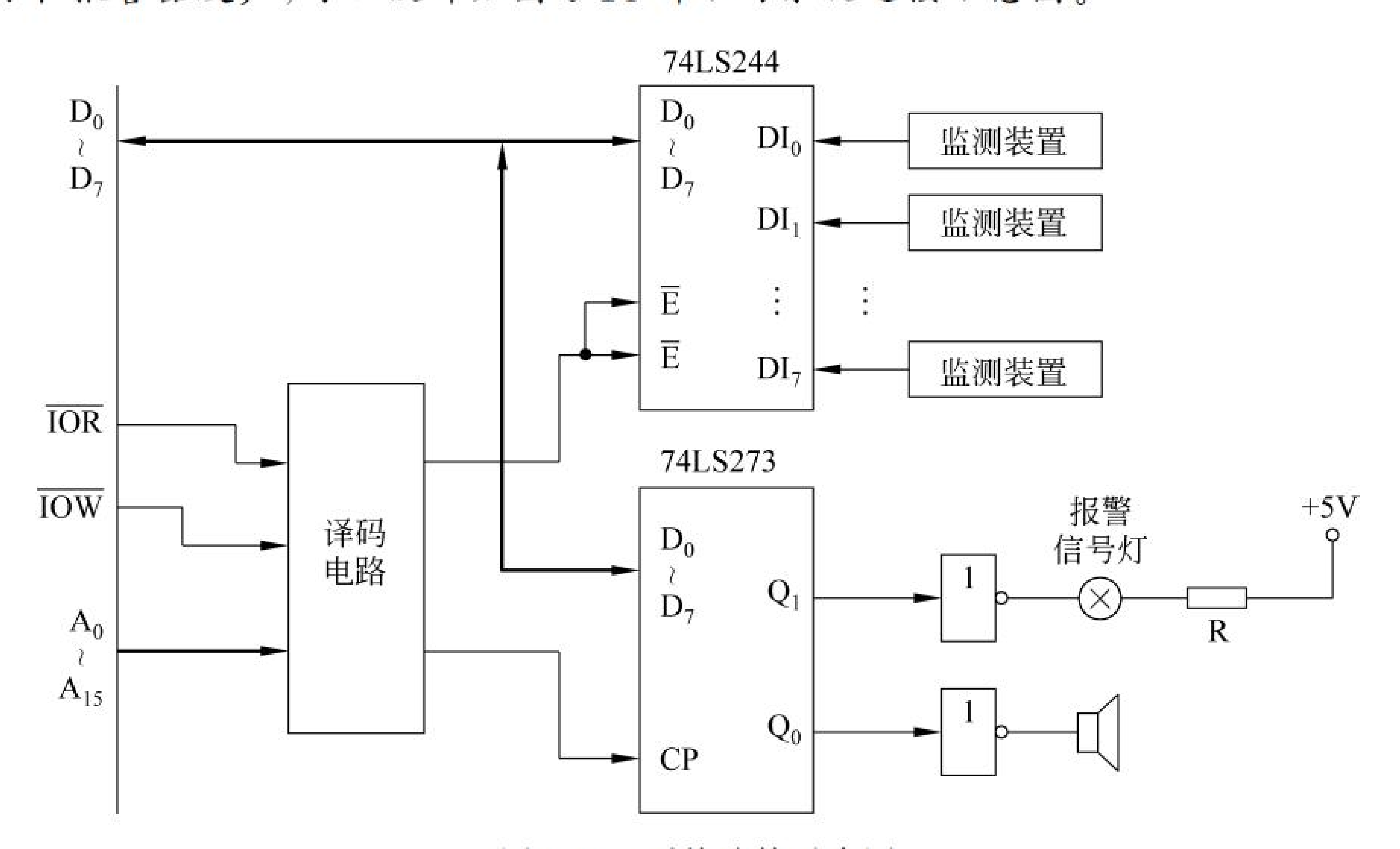


图 6-14 系统连接示意图

现在,可以参照图 6-13 所示的控制流程,设计相应的控制程序了。试一下!

图 6-13 给出了外设利用查询方式进行数据传送的工作过程。但事实上,一个微机系统往往要连接多个外设,这种情况下 CPU 会对外设逐个进行查询,发现哪个外设准备就绪,就对该外设进行数据传送;然后再查询下一外设,依次循环。此时的工作流程如

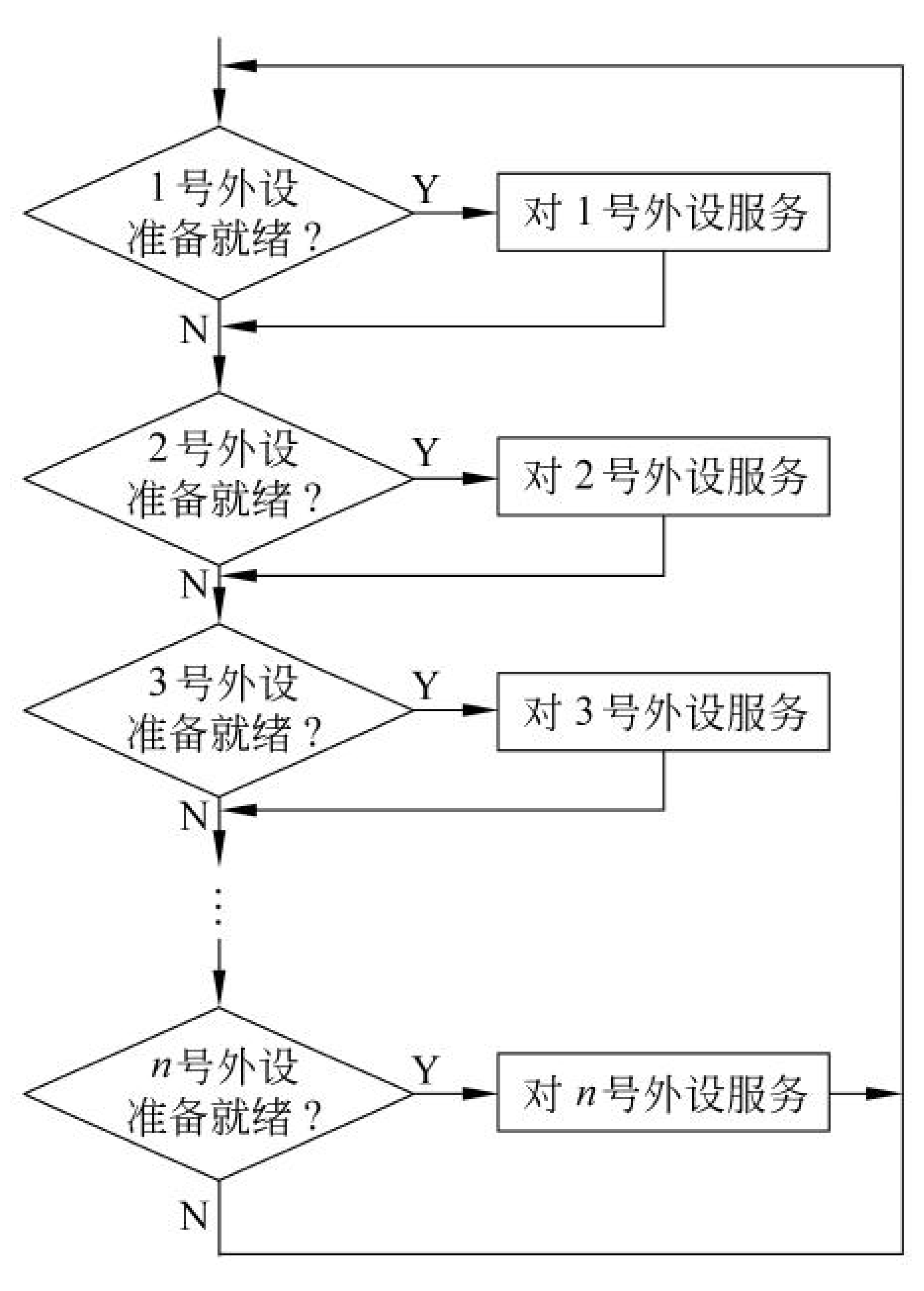


图 6-15　多个外设时的查询工作方式流程图

图 6-15 所示。

由上述可知，利用查询方式进行数据输入输出的过程中，CPU 不能再做别的事，这样大大降低了 CPU 的效率。而且，假如某一外设刚好在查询过之后就处于就绪状态，那么也必须等到 CPU 查询完所有外设，再次查询此外设时，CPU 才能发现它处于就绪状态，然后才能对此外设服务。这使得数据交换的实时性较差，对许多实时性要求较高的外设来说，就有可能丢失数据。

因此，利用查询方式与外设进行数据交换，需要满足以下两点。

(1) 连接到系统的外部设备是简单的、慢速的，且对实时性要求不高。

(2) 连接到同一系统的外设，其工作速度是相近的。如果速度相差过大，可能会造成某些设备的数据丢失。

6.3.3　中断方式

无条件数据传送和查询数据传送方式都是在满足一定条件下采用的。无条件传送适用于慢速外设，其软、硬件都比较简单，但适用范围较窄，且 CPU 与外设不同步时容易出错；而查询方式将大量时间耗费在读取外设状态及进行检测上，真正用于传送数据的时间很少，这降低了 CPU 的效率，并在多个外设的情况下无法对一些外部事件进行实时响应，因此，它也多用于慢速和中速外设。

以上两种输入输出方式都是由 CPU 管理外部设备，在管理的过程中 CPU 不能做别的事情，这对具有多外设且要求实时性较强的计算机控制系统是不适合的。由此就引进了中断的概念，即 CPU 并不主动介入外设的数据传输工作，而是由外部设备在需要进行数据传送时向 CPU 发出中断请求，CPU 在接到请求后若条件允许，则暂停(或中断)正在进行的工作而转去对该外设服务，并在服务结束后回到原来被中断的地方继续原来的工作。这种方式能使 CPU 在没有外设请求时进行原有的工作，有请求时才去处理数据的输入输出，从而提高了 CPU 的利用率。但有一点要注意，CPU 对外设服务结束后要能够回到原来被中断的地方，这就要求在响应中断前必须将返回地址(即中断时 CPU 将要执行的指令的地址)和程序运行状态保存起来，以保证正确返回。这个过程称为断点保护。

利用中断方式进行数据传送，不仅大大提高了 CPU 的效率，还能够对外设的请求作出实时响应。尤其是在外设出现故障、不立即进行处理有可能造成严重后果的情况下，利用中断方式，可以及时做出处理，避免不必要的损失。有关中断的概念、工作原理及中断

源分类等将在本章的 6.4 节仔细讨论。

6.3.4 直接存储器存取方式

虽然采用中断方式能大大提高 CPU 的利用率，但与其他两种方式一样，实际的数据传送过程还是需要 CPU 执行程序来实现，即 CPU 首先将数据从内存（或外设）读到累加器，再写入到接口（或内存）中。因此，以上 3 种方式被统称为程序控制输入输出方式（Programmed Input and Output，PIO）。另外，采用中断方式每进行一次数据传送，都需要保护断点、保护现场等。若再考虑到修改内存地址、判断数据块是否传送完等因素，8088 CPU 通常传送一个字节约需要几十到几百微秒的时间。由此可大致估计出用 PIO 方式的数据传送速率约为每秒几十 KB。这种传送速度对于一些高速外设及批量数据交换（如磁盘与内存的数据交换）来说是不能满足要求的。

对需要高速数据传送的场合，希望外设能够不通过 CPU 而直接与存储器进行信息交换，这就是直接存储器存取（Direct Memory Access，DMA）方式，即通过特殊的硬件电路来控制存储器与外设直接进行数据传送。在这种方式下，CPU 放弃对总线的管理，而由硬件来控制，这个硬件称为 DMA 控制器。典型的 DMA 控制器是 Intel 公司的 8237。下面简单介绍 DMA 控制器的功能及工作过程。

1. DMA 控制器的功能

通常情况下，系统的地址总线、数据总线和一些控制信号，如 $IO/\overline{M}$、$\overline{RD}$、$\overline{WR}$等是由 CPU 管理的，而在 DMA 方式下，DMA 控制器接管这些信号线的控制权，这就要求 DMA 控制器具有以下功能。

（1）收到接口发出的 DMA 请求后，DMA 控制器要向 CPU 发出总线请求信号 HOLD（高电平有效），请求 CPU 放弃总线的控制。

（2）当 CPU 响应请求并发出响应信号 HLDA（高电平有效）后，这时 DMA 控制器要接管总线的控制权，实现对总线的控制。

（3）能向地址总线发出内存地址信息，找到相应单元并能够自动修改其地址计数器。

（4）能向存储器或外设发出读写命令。

（5）能决定传送的字节数，并判断 DMA 传送是否结束。

（6）在 DMA 过程结束后，能向 CPU 发出 DMA 结束信号，将总线控制权交还给 CPU。

2. DMA 控制器的工作过程

DMA 的工作过程大致如下。

（1）当外设准备好，可以进行 DMA 传送时，外设向 DMA 控制器发出 DMA 传送请求信号（DRQ）。

（2）DMA 控制器收到请求后，向 CPU 发出“总线请求”信号 HOLD，表示希望占用总线。

(3) CPU 在完成当前总线周期后会立即对 HOLD 信号进行响应。响应包括两个方面：①CPU 将数据总线、地址总线和相应的控制信号线均置为高阻态，由此放弃对总线的控制权；②CPU 向 DMA 控制器发出“总线响应”信号(HLDA)。

(4) DMA 控制器收到 HLDA 信号后就开始控制总线，并向外设发出 DMA 响应信号 DACK。

(5) DMA 控制器送出地址信号和相应的控制信号，实现外设与内存或内存与内存之间的直接数据传送。例如，在地址总线上发出存储器的地址，向存储器发出写信号 $\overline{MEMW}$，同时向外设发出 I/O 地址、$\overline{IOR}$和 AEN 信号，即可从外设向内存传送一个字节。

(6) DMA 控制器自动修改地址和字节计数器，并据此判断是否需要重复传送操作。规定的数据传送完后，DMA 控制器就撤销发往 CPU 的 HOLD 信号。CPU 检测到 HOLD 失效后，紧接着撤销 HLDA 信号，并在下一时钟周期重新开始控制总线，继续执行原来的程序。

图 6-16 所示的是 DMA 方式中存储器写的总线周期时序，图中 DMA 控制器在 HLDA 有效期间获得总线控制权，在 S_3 周期和 S_4 周期之间插入了一个等待的时钟周期 S_W。在 S_1～S_3 期间，DMAC 送出地址信号和控制信号，选中写入的内存地址单元，将外设提供的有效数据写入规定的内存单元。

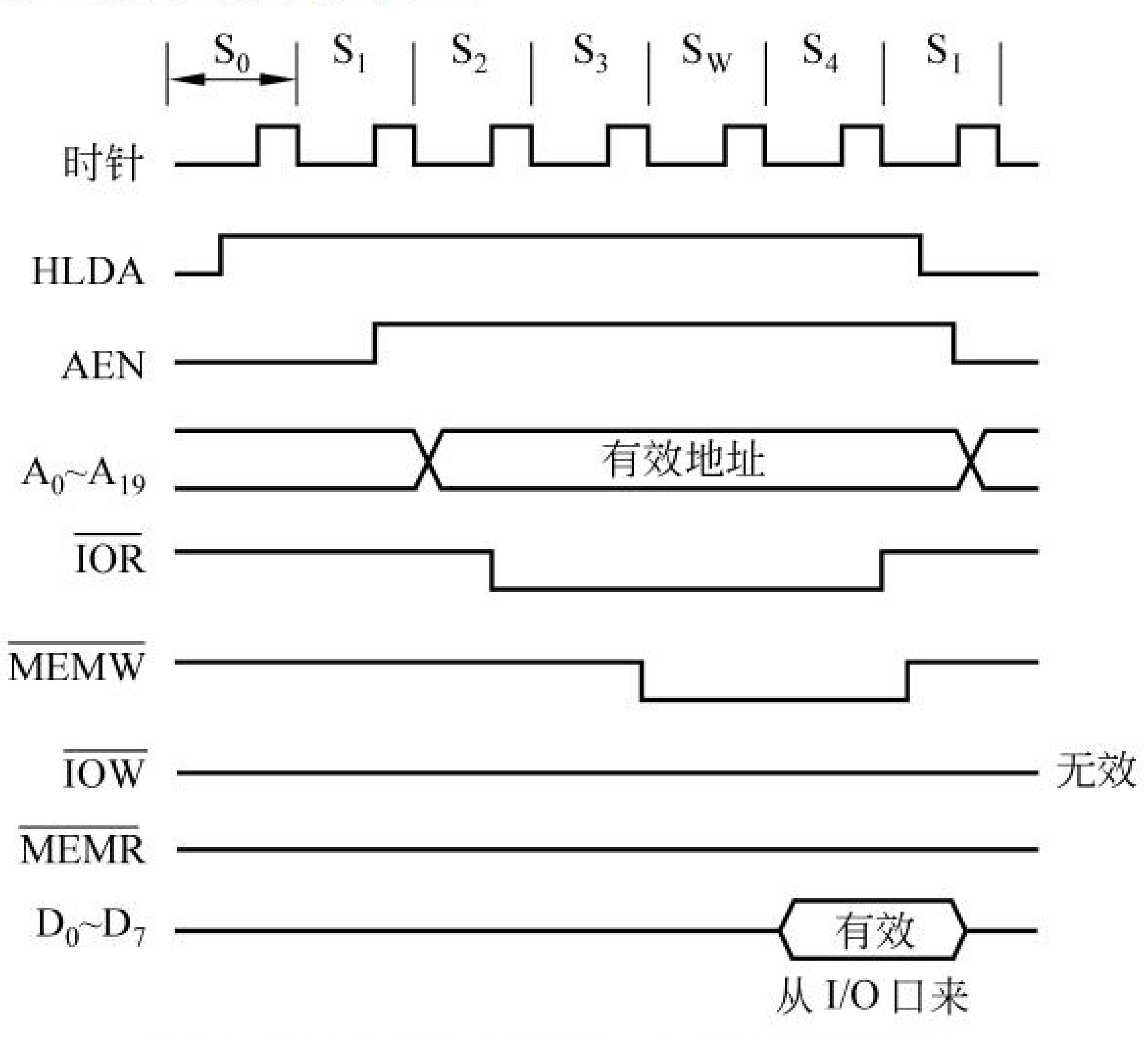


图 6-16　DMA 存储器写的总线周期时序

为了进一步说明 DMA 的传送过程，图 6-17 给出了一个 DMA 存储器写操作的简要原理图。这里要注意两点：①DMA 传送前，CPU 必须告诉 DMA 控制器传送是在哪两个部件之间进行的，传送的内存首地址以及传送的字节数是多少；②在 DMA 传送时，DMA 控制器只负责送出地址及控制信号，而数据传送是直接在接口和内存间进行的，并不经过 DMA 控制器。对于内存与内存间的 DMA 传送，是先用一个 DMA 的存储器读周期将数据由内存读出，放在 DMA 控制器的内部数据暂存器中，再利用一个 DMA 的存储器写周期将该数据写到内存的另一区域。

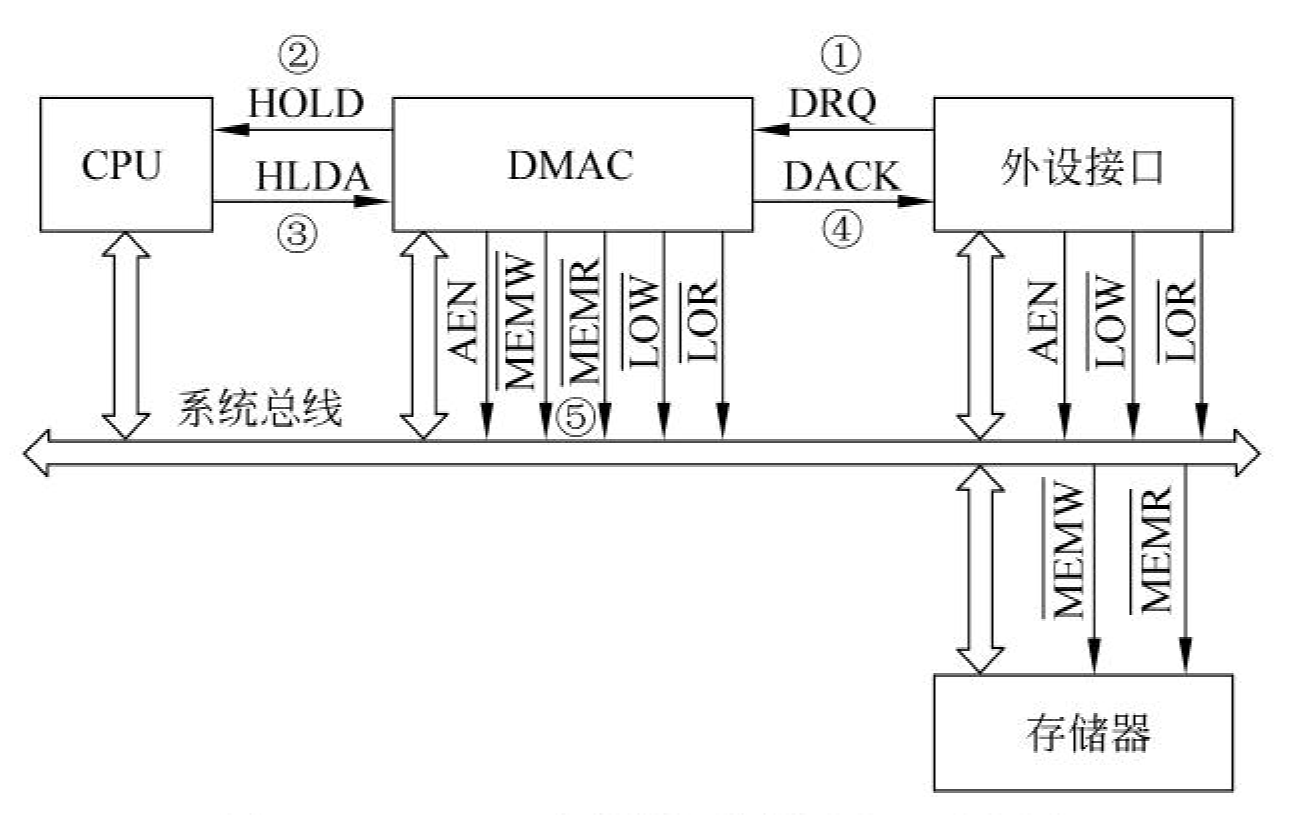


图 6-17 DMA 存储器写操作原理示意图

6.4 中断技术

中断技术在计算机中应用极为广泛，它不仅可用于数据传输、提高数据传输过程中 CPU 的利用率，还可以用来处理一些需要实时响应的事件，例如异常、时钟、掉电、特殊状态等。在操作系统(Operating System，OS)中，还使用中断来进行一些系统级的特殊操作，如虚拟存储器中页面的调入调出等。

6.4.1 中断的基本概念

在微机中，当 CPU 执行程序过程时，由于随机的事件(包括 CPU 内部的和 CPU 外部的事件)引起 CPU 暂时停止正在执行的程序，而转去执行一个用于处理该事件的程序——称为中断服务程序(或中断处理程序)，处理完后又返回被中止的程序断点处继续执行，这一过程就称为中断。

引起中断的事件就称为中断源，即引起中断的原因或来源。中断源可分为两大类：①来自 CPU 内部，称之为内部中断源；②来自 CPU 外部，称之为外部中断源。

内部中断源主要包括：①CPU 执行指令时产生的异常，如被 0 除、溢出、断点、单步操作等；②特殊操作引起的异常，如存储器越界、缺页等；③由程序员安排在程序中的 INT n 软件中断指令。

外部中断源主要包括：①I/O 设备，如键盘、打印机、鼠标等；②数据通道，如磁盘、数据采集装置、网络等；③实时钟，如定时器时间到；④故障源，如掉电、硬件错、存储器奇偶校验错等。

对内部中断来说，中断的控制完全是在 CPU 内部实现的；而对于外部中断，则是利用 CPU 的两条中断输入信号线 INTR 和 NMI 来告诉 CPU 已发生了中断事件。INTR 称为可屏蔽中断输入信号，因为 CPU 能否响应该信号还受到中断允许标志寄存器 IF 的控制。当 IF＝1(中断)时，CPU 在一条指令执行完后对它作出响应；当 IF＝0(关中断)

时,CPU 不予响应,该中断请求被屏蔽。NMI 称为非屏蔽中断请求输入信号,上升沿有效。它不受标志位 IF 的约束,只要 CPU 在正常地执行程序,它就一定会响应 NMI 的请求。

事实上,在日常生活中,“中断”也是很常见的。例如,当你正在看书时,门铃和电话铃同时响了,这时你必须对这两个事件作出反应,并迅速作出判断:是先接电话还是先开门。假如你认为开门比较紧急,就会暂时停止看书(你可能还会在正看的页码处夹上书签)而先去开门,然后去接听电话,这两个事件处理完后,再从原来中断的地方接着看书。

6.4.2 中断处理的一般过程

上述接电话和开门的例子实际就包含了计算机处理中断的 5 个步骤,即中断请求、中断源识别(中断判优)、中断响应、中断处理和中断返回。下面以外部可屏蔽中断为例,简要介绍中断处理过程的 5 个步骤。

1. 中断请求

外设需要 CPU 服务时,首先要发出一个有效的中断请求信号送到 CPU 的中断输入端。中断请求信号分为边沿触发和电平触发。边沿触发指的是 CPU 根据中断请求端上有无从低到高或从高到低的跳变来决定中断请求信号是否有效;电平触发指的是 CPU 根据中断请求端上有无稳定的电平信号(高电平还是低电平取决于 CPU 的设计)来确定中断请求信号是否有效。一般来说,CPU 能够即时予以响应的中断可以采用边沿触发,而不能即时响应的中断则应采用电平触发,否则中断请求信号就会丢失。8088/8086 CPU 的 NMI 为边沿触发,而 INTR 为电平触发。为了保证产生的中断能被 CPU 处理,INTR 中断请求信号应保持到该请求被 CPU 响应为止。CPU 响应后,INTR 信号还应及时撤除,以免造成多次响应。

2. 中断源识别(中断判优)

当系统具有多个中断源时,由于中断产生的随机性,就有可能在某一时刻有两个以上的中断源同时发出中断请求,而 CPU 往往只有一条中断请求线,并且任一时刻只能响应并处理一个中断,这就要求 CPU 能识别出是哪些中断源申请了中断,找出优先级最高的中断源并响应之,在其处理完后,再响应级别较低的中断源的请求。中断请求事件的识别及其优先级的顺序判定就是中断源识别或说中断判优要解决的问题。中断判优的方法分为软件和硬件两种。

1) 软件判优

软件判优是指由软件来安排各中断源的优先级别。软件判优需要相应电路的支持。电路原理图如图 6-18 所示。在电路中,外设的中断请求信号 IRQ 被锁存在中断请求寄存器中,并通过“或”门相“或”后送到 CPU 的 INTR 端。同时把外设的中断请求状态经并行接口输入 CPU。

若某一中断源发出中断请求,中断请求信号经“或”门送到 CPU 的 INTR 引脚

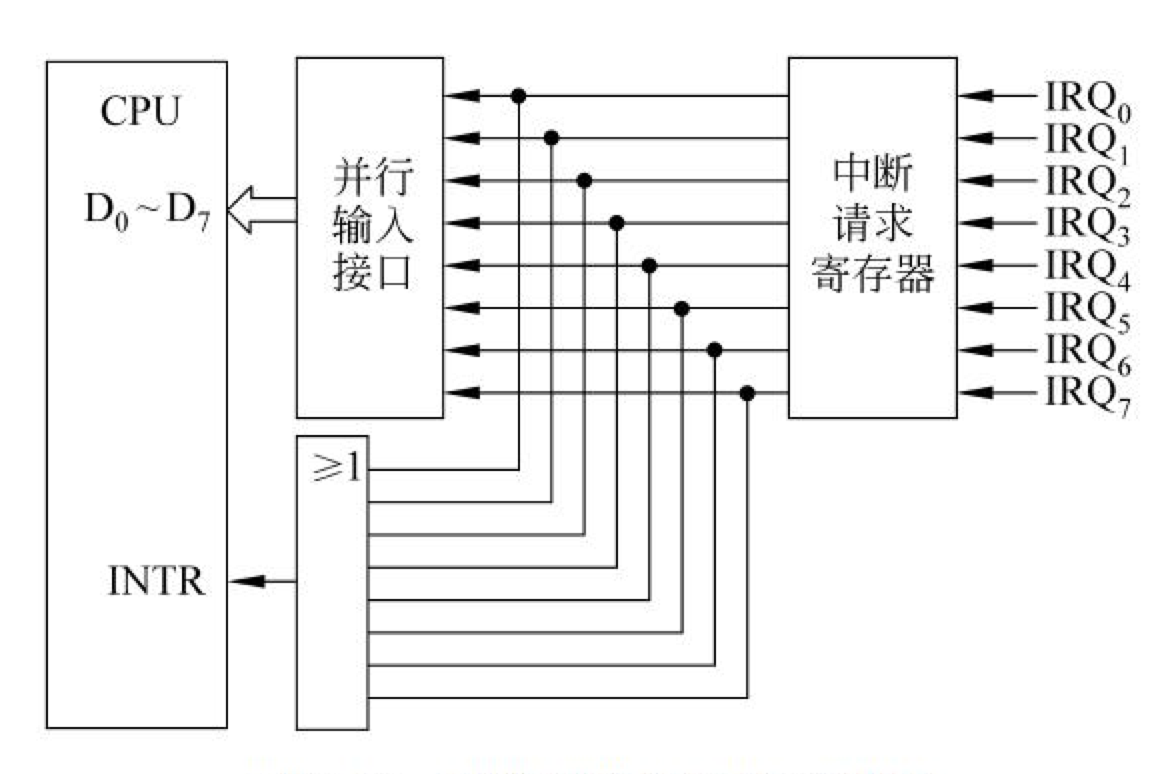


图 6-18 软件判优的电路原理图

上,CPU 响应中断后进入中断处理程序,用软件读取并行端口的中断状态,逐位查询端口的状态,查到哪个中断源有请求就转入哪个中断源的中断服务程序。这里查询的次序就反映了各中断源优先级别的高低,先被查询的中断源优先级别最高,后被查询的中断源优先级依次降低。这种判优方法硬件电路简单、优先权安排灵活,但软件判优所花时间较长,在中断源较多的情况下会影响到中断响应的实时性。硬件判优则可较好地克服这个缺点。

2）硬件判优

硬件判优是指利用专用的硬件电路或中断控制器来安排各中断源的优先级别。硬件判优电路的形式很多,下面介绍两种常用的硬件判优方法。

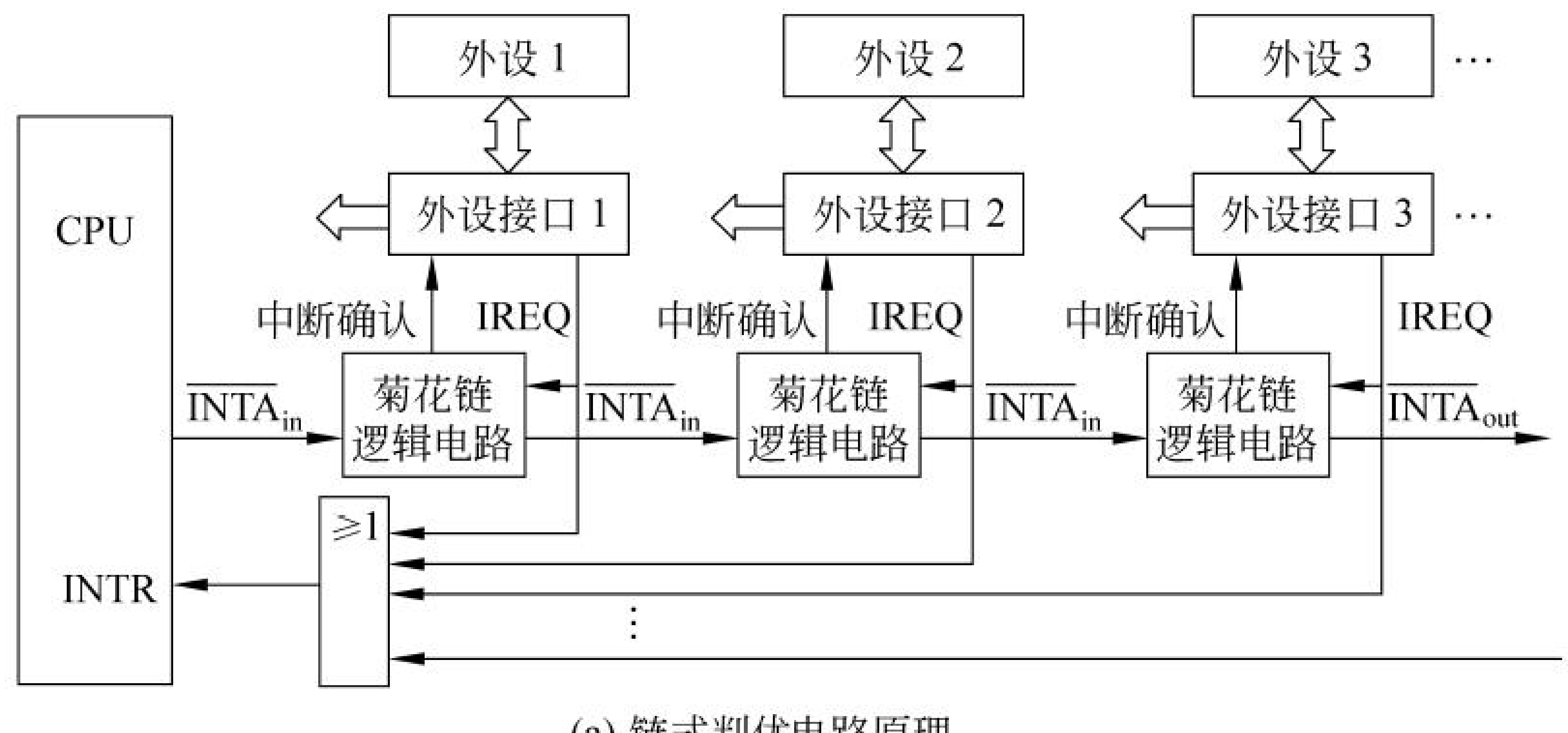
(1) 中断控制器判优。中断控制器判优的核心思想是根据中断向量码(也称中断类型码)来确定中断源。中断向量码是为每一个中断源分配的一个编号,通过该编号可方便地找到与中断源相对应的中断服务程序的入口。

在中断控制器电路中,用一个中断优先级判别器来判别哪个中断请求的优先级最高。当 CPU 响应中断时,将优先级最高的中断源所对应的中断向量码送给 CPU,CPU 根据中断向量码找到相应的中断服务程序入口,对该中断进行处理。

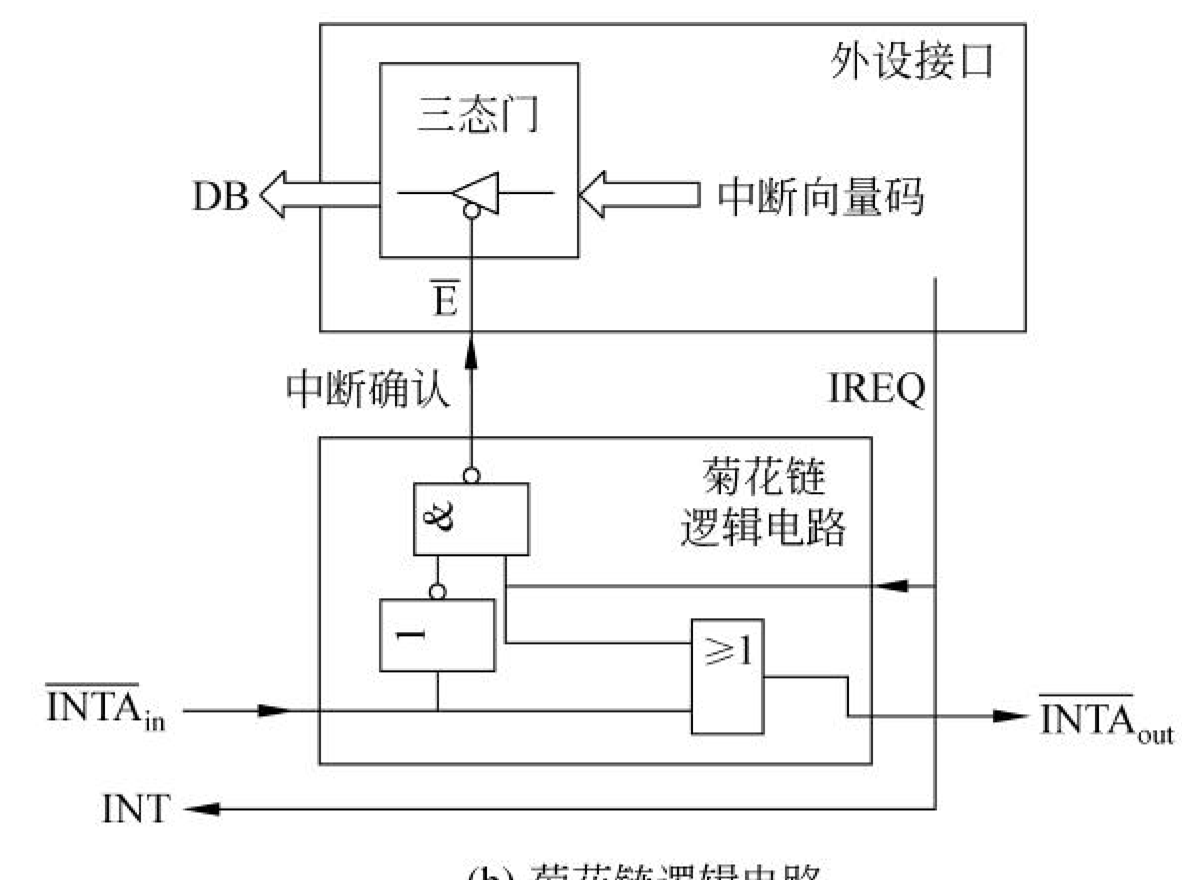
与 8086/8088 CPU 配套的 8259A 芯片是一种可编程的中断控制器,它可对多达 64 级的中断源进行优先级管理,该芯片将在 6.5 节中进行详细介绍。

(2) 链式判优。链式判优的基本思想是将所有的中断源构成一个链(称菊花链),排在链前面的中断源的优先级别高于排在后面的,高优先级别的中断会自动封锁低优先级别的中断。链式优先权排队电路如图 6-19 所示。在电路中,每个外设对应的接口都有一个中断逻辑电路,CPU 响应中断时发出的$\overline{INTA}$信号沿着这些逻辑电路串接成的菊花链从前往后传递。

从图 6-19 中可以看出,当某个外设有中断请求时,CPU 如果允许中断,则会发出$\overline{INTA}$信号。如果菊花链前端的外设没有发出中断请求信号,那么这级中断逻辑电路就会允许中断响应信号$\overline{INTA}$原封不动地向后传递,一直传到发出中断请求的外设。同时,这个外设发出的中断请求会自动对后面设备的中断逻辑电路实现封锁,使$\overline{INTA}$信号不再传到后面的外设(其后的外设的$\overline{INTA}$输入端全部为“1”信号)中。由此可以看出,菊花



(a) 链式判优电路原理



(b) 菊花链逻辑电路

图 6-19　**菊花链中断判优电路**

链电路中各个外设的中断优先权由其在链中的位置决定，处于菊花链前端的比处于链条后端的优先权高。

菊花链前端发出中断请求的外设截获$\overline{\text{INTA}}$信号后就打开三态门，把自己的中断类型码放到数据总线上，CPU 读取该中断类型码，并据此计算出相应的中断服务程序的入口地址，然后转去执行。

当多个外设同时发出中断请求信号时，根据电路分析可知，菊花链中位置靠前的外设将截获$\overline{\text{INTA}}$信号，而排在菊花链中较后位置的外设就收不到$\overline{\text{INTA}}$信号，因而暂时不会被处理。若 CPU 正执行某个中断服务程序时又有级别较高的外设提出中断请求，由于菊花链电路中级别低的外设不能封锁级别高的外设得到中断响应信号，故仍可响应该中断请求。所以，此电路也能实现中断嵌套。

3）中断嵌套问题

中断嵌套类似于子程序嵌套，即高优先级别的中断可以中断低优先级别的中断，出现一层套一层的现象。大部分中断控制电路在解决中断优先级的同时也实现了中断嵌套。中断嵌套的层数一般不受限制，但设计中断程序时要注意留有足够的堆栈空间，因为每一层嵌套都要用堆栈来保护断点，使得堆栈内容不断增加，若堆栈空间过小，中断嵌套层次

较多时就会产生堆栈溢出现象,使程序运行失败。

3. 中断响应

中断优先级确定后,发出中断请求的中断源中优先级最高的请求被送到CPU的中断请求输入引脚上。CPU在每条指令执行的最后一个时钟周期检测中断请求引脚上有无中断请求。但CPU并不是在任何时刻、任何情况下都能对中断请求进行响应。要响应中断请求,必须满足以下4个条件。

(1) 一条指令执行结束。CPU在一条指令执行的最后一个时钟周期对中断请求进行检测,当满足本条件和下述3个条件时,指令执行一结束,CPU即可响应中断。

(2) CPU处于开中断状态。只有在CPU的IF=1,即处于开中断状态时,CPU才有可能响应可屏蔽中断(INTR)请求(对NMI及内部中断无此要求)。

(3) 当前没有发生复位(RESET)、保持(HOLD)、内部中断和非屏蔽中断请求(NMI)。在复位或保持状态时,CPU不工作,不可能响应中断请求;而NMI的优先级比INTR高,当两者同时产生时,CPU会响应NMI而不响应INTR。

(4) 若当前执行的指令是开中断指令(STI)和中断返回指令(IRET),则它们执行完后再执行一条指令,CPU才能响应INTR请求。另外,对前缀指令,如LOCK、REP等,CPU会把它们和它们后面的指令看做一个整体,直到这个整体指令执行完,方可响应INTR请求。

中断响应时,CPU除了要向中断源发出中断响应信号外,还要做下述3项工作。

(1) 保护硬件现场,即FLAGS(PSW)。

(2) 保护断点。将断点的段基地址(CS值)和偏移地址(IP值)压入堆栈,以保证中断结束后能正常返回被中断的程序。

(3) 获得中断服务程序入口。

4. 中断处理

中断处理由中断服务子程序完成。中断服务子程序在形式上与一般的子程序基本相同,区别在于:中断服务子程序只能是远过程(类型为FAR);中断服务子程序要用IRET指令返回被中断的程序。

在中断服务子程序中通常要做以下几项工作。

(1) 保护软件现场:保护软件现场是指把中断服务子程序中要用到的寄存器的原内容压入堆栈保存起来。因为中断的发生是随机性的,若不保护现场,就有可能破坏主程序被中断时的状态,从而造成中断返回后主程序无法正确执行。

(2) 开中断:CPU响应中断时会自动关闭中断(使IF=0)。若进入中断服务子程序后允许中断嵌套,则需用指令开中断(使IF=1),如8086/8088中的STI指令。

(3) 执行中断处理程序:不同的中断,其中断处理程序也各不相同,编程人员可根据中断处理的需要来编写。但中断服务处理程序不宜过长和过于复杂,在中断处理程序中停留的时间越短越好,否则程序运行时既容易出乱,也影响对其他中断源的及时处理。通常的处理方法是:在中断服务子程序中只执行那些必须执行的操作,而其他相关操作可

放到中断服务子程序外去执行(例如放到主程序中)。

(4) 关中断:相应的中断处理指令执行结束后需要关中断,以确保有效地恢复被中断程序的现场。在8086/8088 CPU中,关中断指令为CLI。

(5) 恢复现场:就是把先前保护的现场进行恢复,也即把所保存的有关寄存器内容按压栈的相反顺序从堆栈中弹出,使这些寄存器恢复到中断前的状态。

5. 中断返回

中断返回需执行中断返回指令IRET,其操作正好是CPU硬件在中断响应时自动保护硬件现场和断点的逆过程,即CPU会自动地将堆栈内保存的断点信息和FLAGS弹出到IP、CS和FLAGS中,保证被中断的程序从断点处继续往下执行。

从某个中断源发出中断请求到该中断请求全部处理完成所经过的主要过程的流程图如图6-20所示。

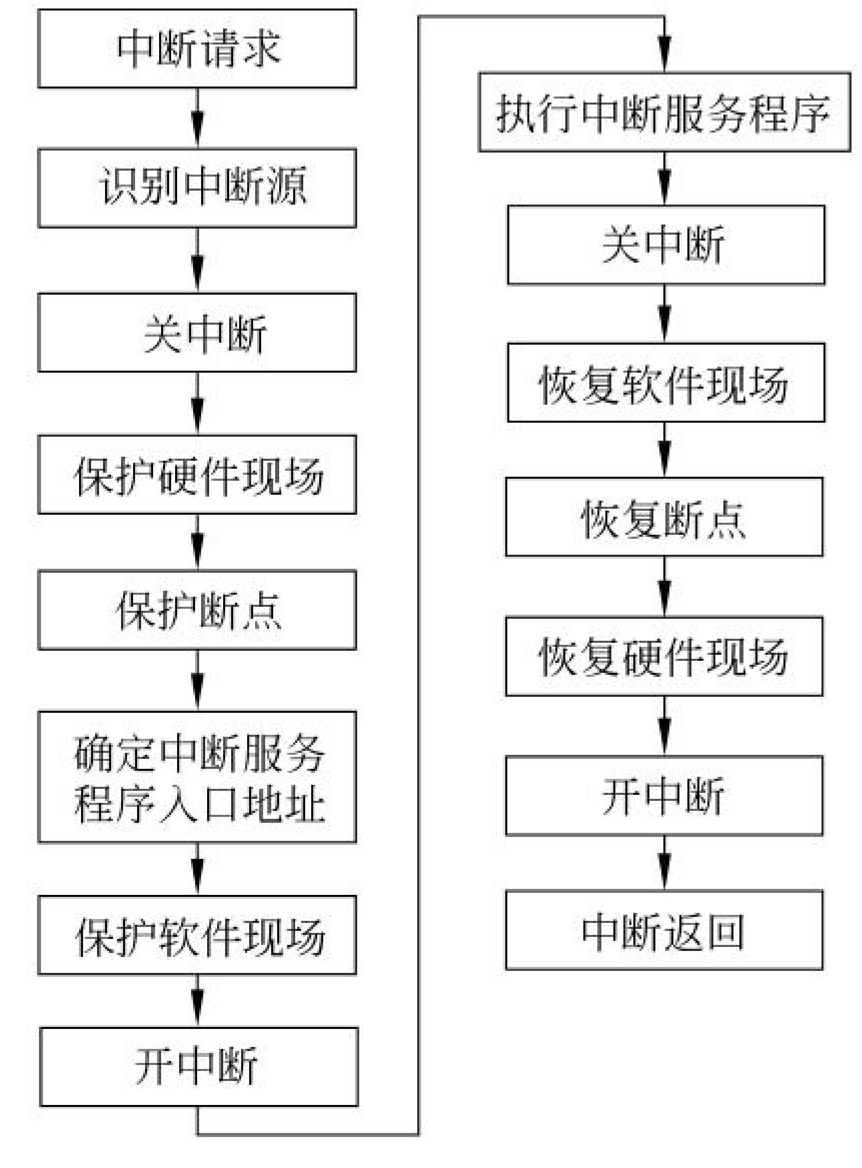


图6-20 中断处理过程流程图

6.4.3 8086/8088中断系统

8086/8088 CPU的中断系统功能很强,使用非常灵活,它可以处理256种不同类型的中断。为了便于识别,8086/8088系统中给每种中断都赋予一个中断类型码(或称中断向量码),编号为0~255。CPU可根据中断类型码的不同来识别不同的中断源。8086/8088系统的中断源可来自CPU外部,称为外部中断;也可以来自CPU内部,称为内部中断,如图6-21所示。

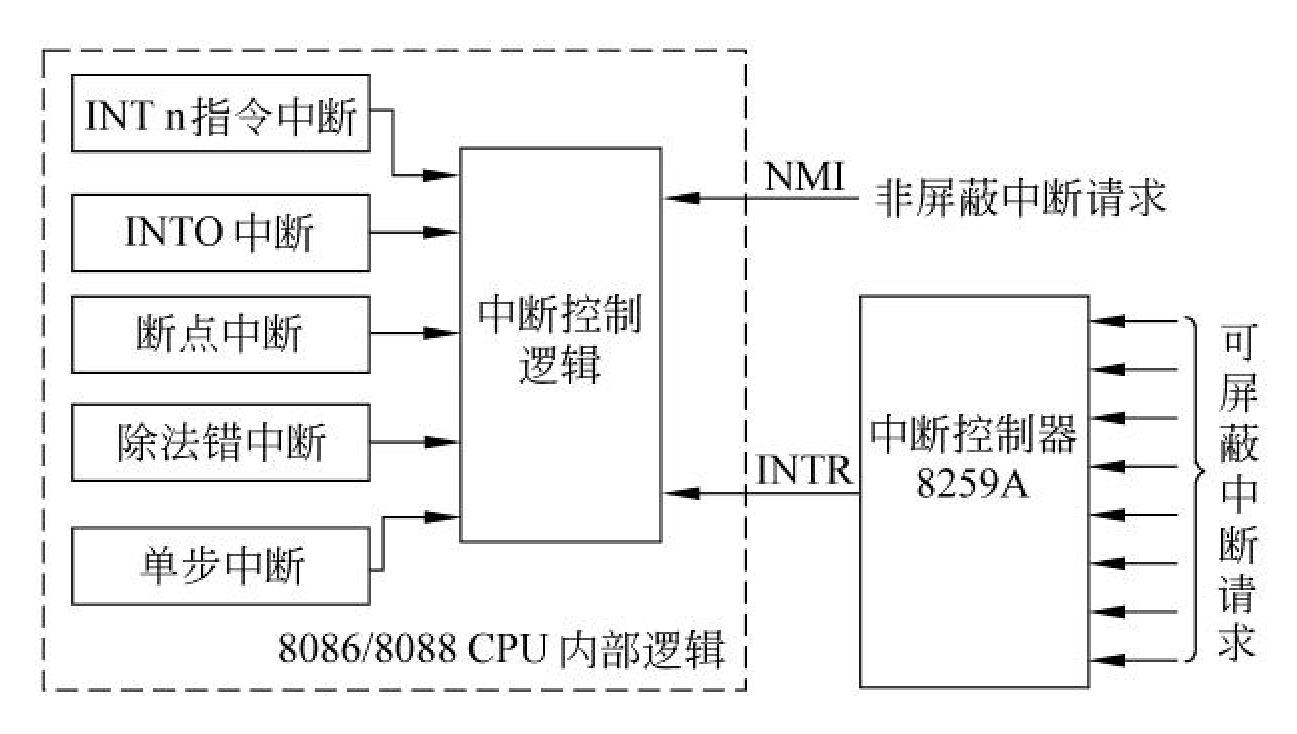


图6-21 8086/8088中断源类型

1. 内部中断

内部中断是 CPU 执行了某条指令或者软件对标志寄存器中某个标志位进行设置而产生的，由于它与外部硬件电路完全无关，故也称其为软件中断。在 8086/8088 CPU 中，内部中断可分为以下 5 种类型。

1）除法出错中断——0 型中断

8086/8088 执行除法指令时，若发现除数为 0 或商超过了结果寄存器所能表示的最大范围，则立即产生一个中断类型码为 0 的中断，该中断称为除法出错中断。该中断的服务处理一般由系统软件进行。

2）单步中断——1 型中断

8086/8088 CPU 的标志寄存器中有一位陷阱标志 TF。CPU 每执行完一条指令都会检查 TF 的状态。若发现 TF＝1，CPU 就产生中断类型码为 1 的中断，使 CPU 转向单步中断的处理程序。单步中断广泛地用于程序的调试，使 CPU 一次执行一条指令，从而能够逐条指令地观察程序运行情况。在程序排错时，单步中断是一种很有效的调试手段。

对单步中断要注意两点：①所有类型的中断在其处理过程中，CPU 都会自动地把状态标志压入堆栈，然后清除 TF 和 IF，因此当 CPU 进入单步中断处理程序时就不再处于单步方式，而以正常方式工作，只有在单步处理结束后，从堆栈中弹出原来的标志，才使 CPU 又回到单步方式；②8086/8088 指令系统中没有设置或清除 TF 标志的指令，但指令系统中的 PUSHF 和 POPF 为程序员提供了置位或复位 TF 的手段。置位和复位 TF 的程序段如下：

```
;置位 TF标志
PUSHF
POP AX
OR AX,0100H                   ;TF置为 1
PUSH AX
POPF

;复位 TF标志
PUSHF
POP AX
AND AX,0FEFFH                 ;TF置为 0
PUSH AX
POPF
```

3）断点中断——3 型中断

8086/8088 指令系统中有一条专用于设置断点的指令，其操作码为单字节 0CCH（助记符为 INT 3）。CPU 执行该指令就会产生一个中断类型码为 3 的中断。INT 3 指令是单字节指令，因而它能很方便地插入到程序的任何地方，专门用于在程序中设置断点来调试程序，它也称为断点中断，插入 INT 3 指令之处便是断点。在断点中断服务子程序

中，可显示有关的寄存器、存储单元等内容，以便程序员分析到断点为止程序运行是否正确。

4）溢出中断——4 型中断

若算术指令的执行结果发生溢出（OF＝1），则执行 INTO 指令后立即产生一个中断类型码为 4 的中断。4 型中断为程序员提供了处理运算溢出的手段，INTO 指令通常和算术指令配合起来使用。

5）用户自定义的软件中断——n 型中断

CPU 执行中断指令 INT n 也会引起内部中断，其中断类型码由指令中的 n 指定。这一类指令统称为软中断指令。除 INT 3 指令（断点中断）外，其余的 INT n 指令的代码为两字节（第一字节为操作码，第二字节为中断类型码）。

实际上，INT n 软中断可以模拟任何类型的中断，在调试那些非 INT n 中断的中断服务子程序时，可以用 INT n 指令来模拟它们发出的中断请求，使原本非常难于调试的中断子程序变得非常简单。

以上所述内部中断的类型码均是固定的或包含在软中断指令中，除单步中断外，其他的内部中断不受 IF 状态标志影响，用于中断处理的中断服务子程序需用户自行编制。

2. 外部中断

外部中断也称为硬件中断，它是由外部硬件或外设接口产生的。8086/8088 CPU 为外部设备提供了两条硬件中断信号线 NMI 和 INTR，非屏蔽中断和可屏蔽中断请求信号分别从这两个引脚送入 CPU。

1）非屏蔽中断

非屏蔽中断由 NMI 引脚上出现的上升沿触发，它不受中断允许标志 IF 的限制，其中断类型码固定为 2。

CPU 接收到非屏蔽中断请求信号后，不管当前正在做什么事，都会在执行完当前指令后立即响应中断请求而进入相应的中断处理。非屏蔽中断通常用来处理系统中出现的重大故障或紧急情况，如系统掉电处理、紧急停机处理等。在 PC 中，若系统板上的存储器或 I/O 通道上产生了奇偶校验错以及 8087 数学协处理器产生异常都会引起一个 NMI 中断。

2）可屏蔽中断

绝大多数外部设备提出的中断请求都是可屏蔽中断，可屏蔽中断的中断请求信号从 CPU 的 INTR 端引入，高电平有效。可屏蔽中断受中断允许标志位 IF 的约束，只有当 IF＝1 时，CPU 才会响应 INTR 请求。如果 IF＝0，即使中断源有中断请求，CPU 也不会响应，这种情况称为中断被屏蔽。在 PC 中，外部设备的中断请求是通过中断控制器 8259A 来进行统一管理的，由 8259A 决定是否允许一个外设向 CPU 发出中断请求。IBM PC 中的可屏蔽中断的中断类型码为 8～15（08H～0FH），80286 以后的微机还包括 112～119（70H～77H）。

3. 中断向量表

在 8086/8088 CPU 中断系统中，无论是外部中断还是内部中断，每个中断源都有一个与它相对应的中断类型码，它是中断源在系统中的“身份证”。中断类型码长度为一个字节，故 8086/8088 最多允许处理 256 种类型的中断（中断类型码为 0～255）。CPU 在响应中断时，通过得到的中断类型码来判断是哪个中断源提出了中断请求。

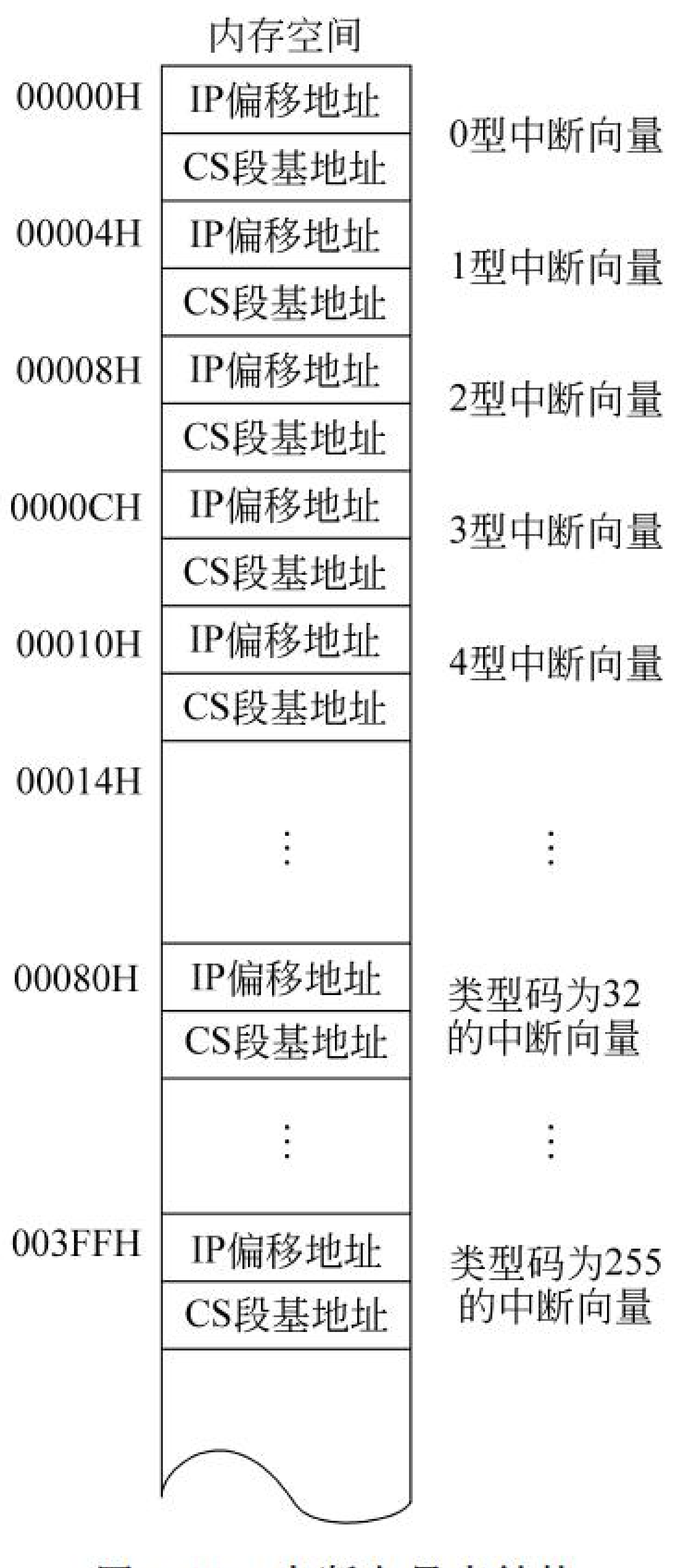


图 6-22　中断向量表结构

为了能够根据所得到的中断类型码来找到中断服务子程序的首地址，8086/8088 系统规定所有中断服务子程序的首地址都必须放在一个称为中断向量表的表格中（类似于 C 语言中的指针数组）。中断向量表位于内存中最低的 1KB（即内存中 00000H～003FFH 区域），共有 256 个表项，用以存放 256 个中断向量（即 256 个中断服务子程序的入口地址）。每个中断向量（表项）占 4 个字节，其中低位字（2 个字节）存放中断服务子程序入口地址的偏移量，高位字存放中断服务子程序入口地址的段地址。按照中断类型码的大小，对应的中断向量在中断向量表中有规则地顺序存放，如图 6-22 所示。

根据中断向量表的格式，只要知道了中断类型码 n 就可以找到所对应的中断向量在表中的位置。中断向量在中断向量表中的存放位置（地址）可由下式计算得到：

$$中断向量在表中的存放地址 = n \times 4$$

例如，中断类型码为 21H 的中断，其中断向量存放在 0000:0084H（4×21H＝84H）开始的 4 个字节单元中。

计算出中断向量地址后，只要取 $4n$ 和 $4n+1$ 单元的内容装入 IP，取 $4n+2$ 和 $4n+3$ 单元的内容装入 CS，即可转入中断服务子程序。

需要注意的是，在 80386 以后的微机中，由于虚存及保护方式的出现，中断向量表不再是固定放在 00000H～003FFH 区域中（中断向量表的名字也改为中断描述符表 IDT），而是可以位于内存的任意区域，表的首地址放在 CPU 内部的 IDT 基址寄存器中。每个表项也从 4 个字节增加到了 8 个字节，包括 2 字节的选择器、4 字节的偏移量和 2 字节的其他属性。

4. 8086/8088 CPU 的中断响应过程

8086/8088 对不同类型中断的响应过程不同，主要区别在于如何获得相应的中断类型码。

1）内部中断响应过程

CPU在执行内部中断时，没有中断响应周期。对于除法溢出、单步、断点和溢出中断，中断类型码是自动形成的，而对于INT n指令，其中断类型码由INT n指令中给定的 n 决定。获得中断类型码以后的处理过程如下。

（1）将类型码乘4，计算出中断向量的地址。

（2）硬件现场保护，即将标志寄存器FLAGS压入堆栈，以保护当前指令执行结果的特征。

（3）清除IF和TF标志，屏蔽新的INTR中断和单步中断。

（4）保存断点，即把断点处的IP和CS值压入堆栈，先压入CS值，再压入IP值。

（5）根据(1)计算出来的地址从中断向量表中取出中断服务子程序的入口地址（段和偏移），分别送至CS和IP中。

（6）转入中断服务子程序执行。

进入中断服务子程序后，首先要保护在中断服务子程序中要使用的寄存器内容，然后进行相应的中断处理，在中断返回前恢复保护的寄存器内容，最后执行中断返回指令IRET。IRET的执行将使CPU按次序恢复断点处的IP、CS和标志寄存器，从而使程序返回到断点处继续执行。

内部中断具有如下一些特点。

（1）中断由CPU内部引起，中断类型码的获得与外部无关，CPU不需要执行中断响应周期去获得中断类型码。

（2）除单步中断外，内部中断无法用软件禁止，不受中断允许标志IF的影响。

（3）内部中断何时发生是可以预测的，这类似于子程序调用。

2）外部中断响应过程

（1）非屏蔽中断响应。NMI中断不受IF标志的影响，也不用外部接口给出中断类型码，CPU响应NMI中断时也没有中断响应周期。CPU会自动按中断类型码2来计算中断向量的地址，其后的中断处理过程和内部中断一样。

（2）可屏蔽中断响应。当INTR信号有效时，如果中断允许标志IF＝1，则CPU就会在当前指令执行完毕后，产生两个连续的中断响应总线周期。在第一个中断响应总线周期，CPU将地址/数据总线置高阻，发出第一个中断响应信号 $\overline{\text{INTA}}$ 给8259A中断控制器，表示CPU响应此中断请求，禁止来自其他总线控制器的总线请求。在最大模式时，CPU还要启动LOCK信号，通知总线仲裁器8289，使系统中其他处理器不能访问总线。在第二个中断响应总线周期，CPU送出第二个 $\overline{\text{INTA}}$ 信号，该信号通知8259A中断控制器将相应中断请求的中断类型码放到数据总线上供CPU读取。CPU读取中断类型码 n 后的中断处理过程也和内部中断一样。图6-23给出了8086/8088对INTR的中断响应时序。

以上所述的软件中断、单步中断、断点中断、非屏蔽中断和可屏蔽中断，它们的优先级是由8086/8088 CPU识别中断的前后顺序来决定的。在当前指令执行完后，CPU首先自动查询在指令执行过程中是否有除法出错中断、溢出中断和INT n中断发生，然后查询NMI和INTR，最后查询单步中断。8086/8088中断响应和中断处理流程如图6-24所示。

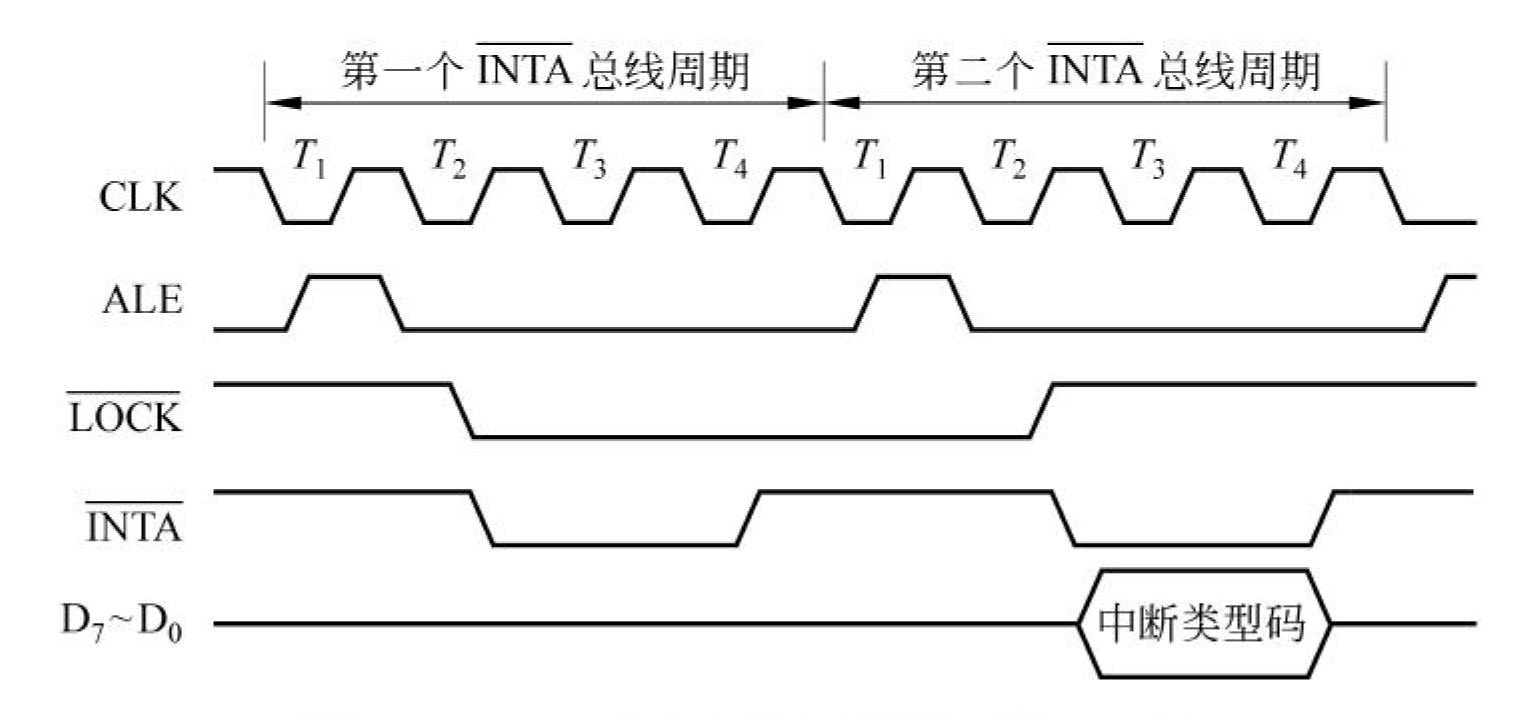


图 6-23　8086/8088 对 INTR 的中断响应时序

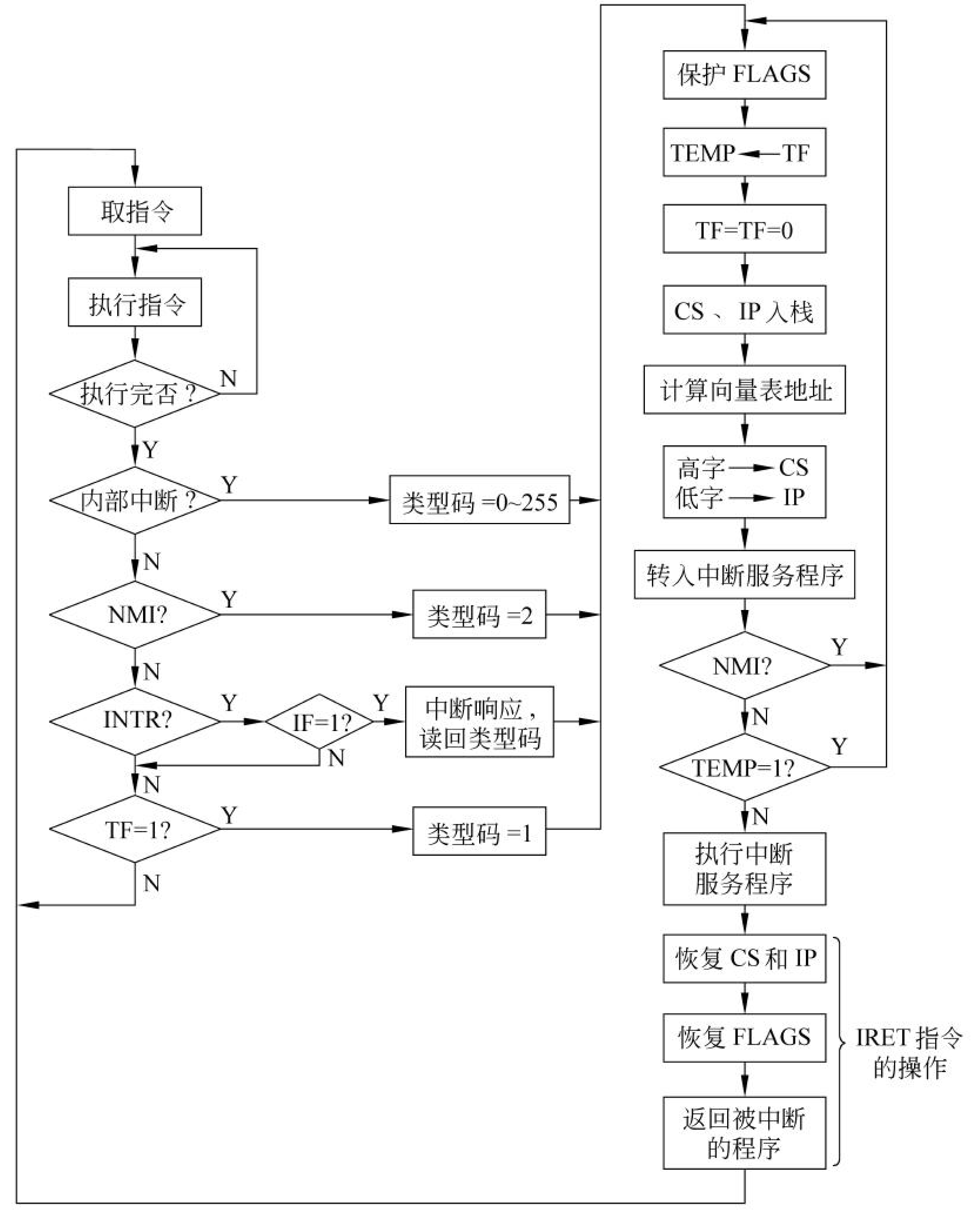


图 6-24　中断响应和中断处理流程

6.5 可编程中断控制器 8259A

8259A 是 Intel 公司生产的专为 8086/8088 CPU 配套的可编程中断控制器(Programmable Interrupt Controller,PIC),用于对 8086/8088 系统中的可屏蔽中断进行管理。8259A 可对 8 个中断源实现优先级控制,多片 8259A 通过级联还可扩展至对 64 个中断源实现优先级控制。8259A 可以根据不同的中断源向 CPU 提供不同的中断类型码,还可根据需要对中断源进行中断屏蔽。8259A 有多种工作方式,可以通过编程来选择,以适应不同的应用场合。

6.5.1 8259A 的引线及内部结构

1. 8259A 的外部引线

8259A 采用 28 引脚双列直插式封装,其外部引线定义如图 6-25 所示。

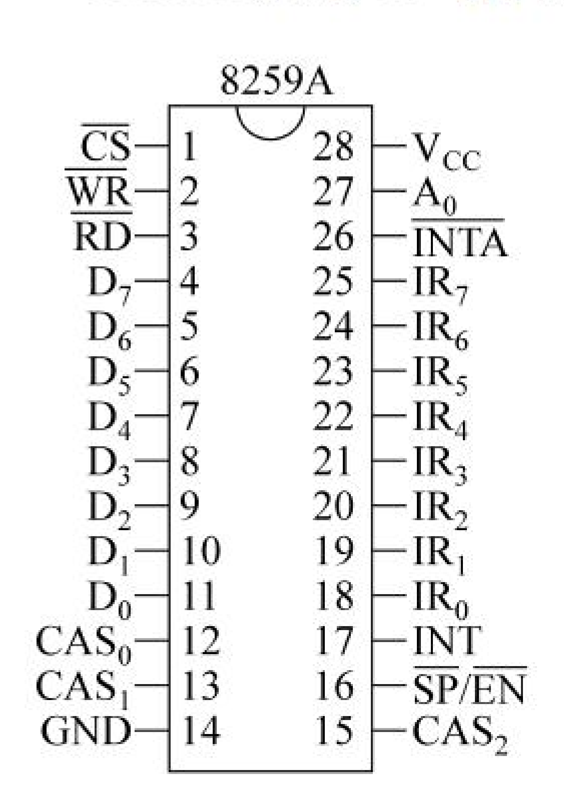


图 6-25 8259A 引线图

(1) D_0～D_7 为双向数据线,与系统的数据总线相连。编程时控制字、命令字由此写入;中断响应时,中断向量码由此送给 CPU。

(2) $\overline{WR}$、$\overline{RD}$为写和读控制信号,与系统总线的$\overline{IOW}$、$\overline{IOR}$相连接。

(3) $\overline{CS}$为片选信号,当$\overline{CS}$为低电平时,8259A 被选中,CPU 才能对它进行读写操作。此引脚连到系统的 I/O 译码器输出,由此确定 8259A 在系统 I/O 地址空间的基地址。

(4) A_0 是 8259A 内部寄存器的选择信号。它与$\overline{CS}$、$\overline{WR}$、$\overline{RD}$信号相配合,对不同的内部寄存器进行读写。使用中,通常接地址总线的某一位,例如 A_1 或 A_0 等。

(5) INT 为 8259A 的中断请求输出信号,可直接接到 CPU 的 INTR 输入端。

(6) $\overline{INTA}$为中断响应输入信号。在中断响应过程中 CPU 的中断响应信号由此端进入 8259A。

(7) CAS_0～CAS_2 为级联控制线。当多片 8259A 级联工作时,其中一片为主控芯片,其他均为从属芯片。对于主片 8259A,其 CAS_0～CAS_2 为输出;对各从片 8259A,它们的 CAS_0～CAS_2 为输入。主片的 CAS_0～CAS_2 与从片的 CAS_0～CAS_2 对应相连。当某从片 8259A 提出中断请求时,主片 8259A 通过 CAS_0～CAS_2 送出相应的编码给从片,使从片的中断被允许。

(8) $\overline{SP}/\overline{EN}$为双功能引线。当 8259A 工作在缓冲模式时,它为输出,用以控制缓冲器的传送方向。当数据从 CPU 送往 8259A 时,$\overline{SP}/\overline{EN}$输出为高电平;当数据从 8259A 送往 CPU 时,$\overline{SP}/\overline{EN}$输出为低电平。在 8259A 工作在非缓冲模式时,它为输入,用于指

定 8259A 是主片还是从片。$\overline{SP}$=1 的 8259A 为主片，$\overline{SP}$=0 的 8259A 为从片。只有一个 8259A 时，它应接高电平。

(9) IR_0～IR_7 为中断请求输入信号，与外设的中断请求线相连。上升沿或高电平(可通过编程设定)时表示有中断请求到达。

2. 8259A 的内部结构

8259A 内部结构如图 6-26 所示。它由中断请求寄存器 IRR(Interrupt Request Register)、中断服务寄存器 ISR(Interrupt Service Register)、中断屏蔽寄存器 IMR(Interrupt Mask Register)、中断判优电路、数据总线缓冲器、读写电路、控制逻辑和级联缓冲/比较器组成。

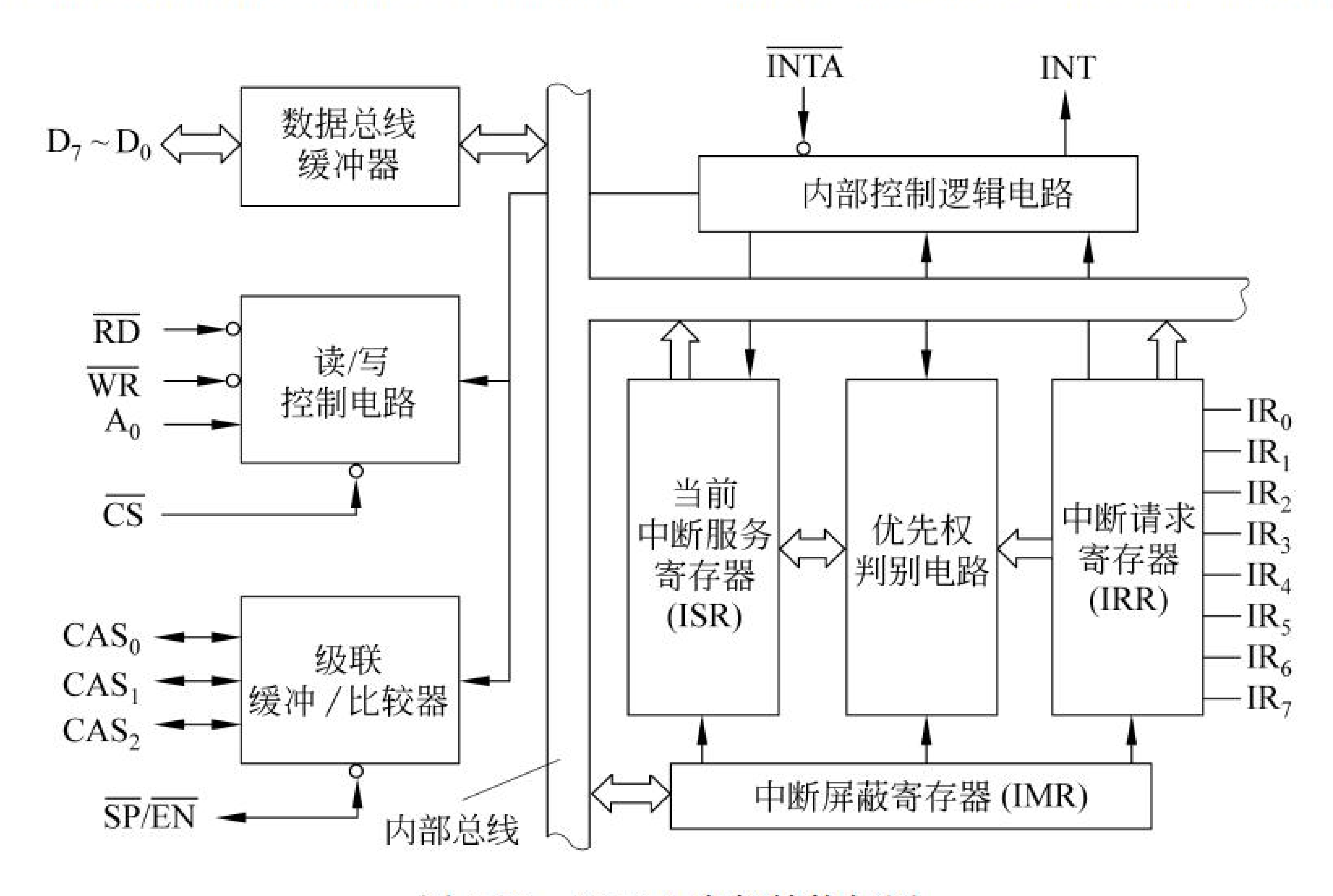


图 6-26 8259A 内部结构框图

1) 中断请求寄存器 IRR

IRR 保存从 IR_0～IR_7 来的中断请求信号。某一位为 1 表示相应引脚上有中断请求信号。该中断请求信号至少应保持到该请求被响应为止。中断响应后，该 IR 输入线上的请求信号应撤销。否则，在中断处理完结后，该 IR 线上的高电平可能会引起又一次中断服务。

2) 中断服务寄存器 ISR

ISR 用于保存所有正在服务的中断源。它是 8 位的寄存器(IS_0～IS_7 分别对应 IR_0～IR_7)。在中断响应时，判优电路把发出中断请求的中断源中优先级最高的中断源所对应的位置 1，以表示该中断请求正在处理中。ISR 的某一位 IS_i 置 1 可阻止与它同级及更低优先级的请求被响应，但不阻止比它优先级高的中断请求被响应，即允许中断嵌套。所以，ISR 中可能有不止一位被置 1。当 8259A 收到"中断结束"(End Of Interrupt，EOI)命令时，ISR 相应位会被清除。对自动 EOI 操作(Automatic EOI，AEOI)，ISR 寄存器中刚被置 1 的位在中断响应结束时自动复位。

3) 中断屏蔽寄存器 IMR

IMR 用于存放中断屏蔽字，它的每一位分别与 IR_7～IR_0 相对应。其中为 1 的位所

对应的中断请求输入将被屏蔽，为 0 的位所对应的中断请求输入不受影响。

4）中断判优电路

中断判优电路监测从 IRR、ISR 和 IMR 来的输入，并确定是否应向 CPU 发出中断请求。在中断响应时，它要确定 ISR 寄存器哪一位应置 1，并将相应的中断类型码送给 CPU。在 EOI 命令时，它要决定 ISR 寄存器哪一位应复位。

6.5.2 8259A 的工作过程

当系统通电后，首先应对 8259A 初始化，也就是由 CPU 执行一段程序，向 8259A 写入若干控制字，使其处于指定的工作方式。当初始化完成后，8259A 就处于就绪状态，随时可接受外设送来的中断请求信号。当外设发出中断请求后，8259A 对外部中断请求的处理过程如下。

(1) 若有一条或若干条中断请求输入线（IR_0～IR_7）上的中断请求信号有效，则 IRR 的相应位置 1。

(2) 若中断请求线中至少有一条是中断未被屏蔽的，则 8259A 由 INT 引脚向 CPU 发出中断请求信号 INTR。

(3) 若 CPU 是处于开中断状态，则在当前指令执行完以后，CPU 用 $\overline{INTA}$ 信号作为对 INTR 的响应。

(4) 8259A 在接收到 CPU 发出的第一个 $\overline{INTA}$ 脉冲后，使最高优先权的 ISR 位置 1，并使相应的 IRR 位复位。

(5) 在第二个中断响应总线周期中，CPU 再输出一个 $\overline{INTA}$ 脉冲，这时 8259A 就把刚才选定的中断源所对应的 8 位中断类型码放到数据总线上。CPU 读取该中断类型码并乘以 4，就可以从中断向量表中取出中断服务子程序的入口地址并转去执行。

(6) 若 8259A 工作在自动中断结束 AEOI 方式，在第二个 $\overline{INTA}$ 脉冲结束时，就会使中断源所对应的 ISR 中的相应位复位。对于非自动中断结束方式，则由 CPU 在中断服务子程序结束时向 8259A 写入 EOI 命令，才能使 ISR 中的相应位复位。

6.5.3 8259A 的工作方式

8259A 具有非常灵活的中断管理方式，可满足用户各种不同的要求，并且这些工作方式都可以通过编程来设置(怎样编程后面会逐步介绍)。由于工作方式较多，因此使用户感到 8259A 的编程和使用不太容易掌握。为此，在讲述 8259A 的编程之前先对 8259A 的工作方式分类进行简单介绍。

1. 中断优先方式与中断嵌套

1）中断优先方式

为了满足实际应用的需要，8259A 提供了两类优先级控制方式：固定优先级和循环优先级方式。

(1) 固定优先级方式。在这种方式下,只要不重新设置优先级别,各中断请求的中断优先级就是固定不变的。8259A 加电后就处于这种方式,刚加电时,默认 IR_0 优先级最高(0 级为最高级),IR_7 优先级最低(7 级为最低级),这种优先顺序也可通过程序予以改变,使它按另外一种顺序排列。图 6-27 给出了两种固定优先级的顺序。

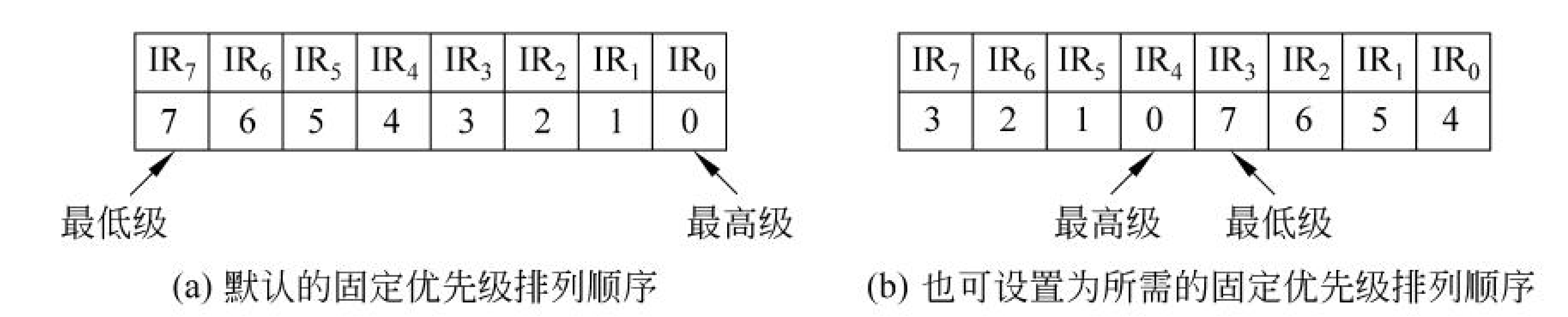


图 6-27 固定优先级方式

(2) 循环优先级方式。在实际应用中,许多中断源的优先权级别是一样的,若采用固定优先级,则低级别中断源的中断请求有可能总是得不到服务。解决的方法是使这些中断源轮流处于最高优先级。这就是自动中断循环优先级方式。

在循环优先级方式中,优先级顺序是变化的。一个中断源得到中断服务以后,它的优先级自动降为最低,原来比它低一级的中断则为最高级,依次排列。例如,若初始优先级从高到低依次为 IR_0、IR_1、IR_2、…、IR_7,此时如果 IR_4 和 IR_6 有中断请求,则先处理 IR_4。在 IR_4 被服务以后,IR_4 自动降为最低优先级,IR_5 成为最高优先级,这时中断源的优先级顺序变为 IR_5、IR_6、IR_7、IR_0、IR_1、IR_2、IR_3、IR_4。

2) 中断嵌套

无论是固定优先级方式还是自动循环优先级方式,它们都允许中断嵌套,即允许更高优先级的中断可以打断当前的中断处理过程。8259A 允许两种中断嵌套方式。

(1) 普通全嵌套方式。普通全嵌套方式是 8259A 最常用的工作方式,简称为全嵌套方式。当 CPU 响应中断时,8259A 将申请中断的中断源中优先权最高的那个中断源在 ISR 中的相应位置 1,并且把它的中断类型码送到数据总线,在此中断源的中断服务子程序完成之前,与它同级或优先权更低的中断源的申请就被屏蔽,只有优先权比它高的中断源的申请才被允许。

(2) 特殊全嵌套方式。特殊全嵌套方式和普通全嵌套方式的差别在于:在特殊全嵌套方式下,当处理某一级中断时,如果有同级的中断请求,8259A 也会给予响应,从而实现一个中断处理过程能被另一个具有同等级别的中断请求所打断。

特殊全嵌套方式一般用在 8259A 级联的系统中。在这种情况下,只有主片 8259A 允许编程为特殊全嵌套方式。这样,当来自某一从片的中断请求正在处理时,主片除对来自优先级较高的本片上其他 IR 引脚上的中断请求进行开放外,同时对来自同一从片的较高优先级请求也会开放。这样可以使从片上优先级别更高的中断得到响应,如图 6-28 所示。

另外,在特殊全嵌套方式中,在中断结束时,应通过软件检查刚结束的中断是否是从片的唯一中断,方法是:先向从片发一正常结束中断命令 EOI,然后读 ISR 内容。若为 0 表示只有一个中断服务,这时再向主片发一个 EOI 命令;否则,说明该从片有两个

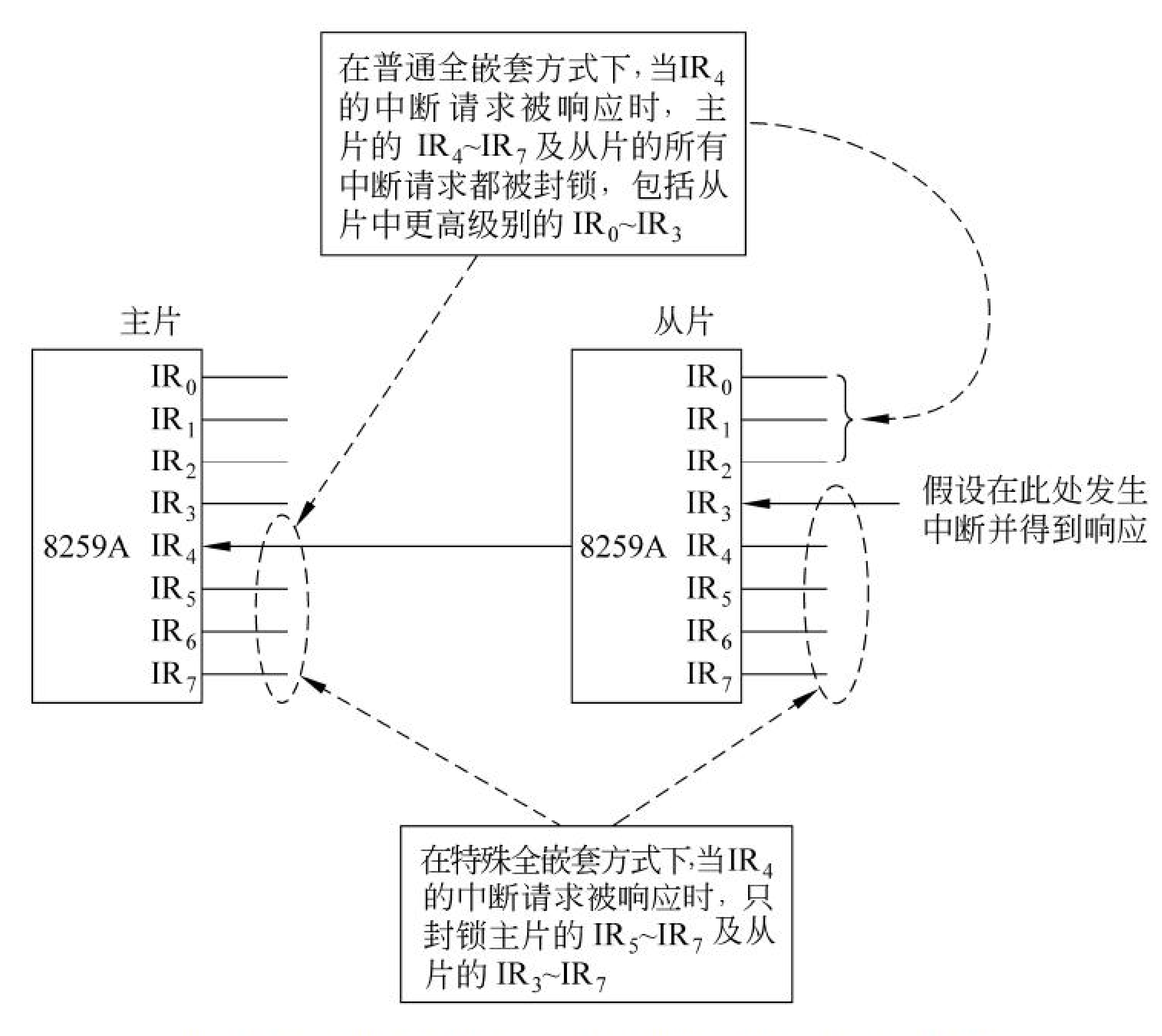


图 6-28　普通全嵌套方式与特殊全嵌套方式的区别

以上中断，则不应向主片发 EOI 命令，待该从片中断服务全部结束后，再发送 EOI 命令给主片。

2. 中断结束处理方式

不管用哪种优先权方式工作，当一个中断请求 IR_i 得到响应时，8259A 都会将中断服务寄存器 ISR 中相应位 IS_i 置 1。而当中断服务程序结束时，则必须将该 IS_i 位清零。否则，8259A 的中断控制功能就会不正常。这个使 IS_i 位复位的动作就是中断结束处理。注意，这里的中断结束是指 8259A 结束中断的处理，而不是 CPU 结束执行中断服务子程序。

8259A 分自动中断结束方式和非自动中断结束方式，而非自动中断结束方式又分为正常（一般）中断结束方式和特殊中断结束方式。

1）自动中断结束方式

若采用自动中断结束方式（AEOI），则在第二个中断响应周期 $\overline{INTA}$ 信号的后沿，8259A 将自动把中断服务寄存器 ISR 中的对应位清除。这样，尽管系统正在为某个设备进行中断服务，但对 8259A 来说，中断服务寄存器中却没有保留正在服务的中断的状态。所以对 8259A 来说，好像中断服务已经结束了一样。这种最简单的中断结束方式只能用于没有中断嵌套的情况。

2）正常中断结束方式

正常中断结束方式配合全嵌套优先权工作方式使用。当 CPU 用输出指令向 8259A 发出正常中断结束 EOI 命令时，8259A 就会把 ISR 中已置 1 的位中的最高位复位。因为

在全嵌套方式中，置1的最高ISR位对应了最后一次被响应的和被处理的中断，也就是当前正在处理的中断，所以，把已置1的位中最高的ISR位复位相当于结束了当前正在处理的中断。

3）特殊中断结束方式

在非全嵌套方式下，由于中断优先级不断改变，无法确知当前正在处理的是哪一级中断，这时就要采用特殊中断结束方式(SEOI)。这种方式反映在程序中就是要发一条特殊中断结束命令，这个命令指出了要清除ISR中的哪一位。

有一点要注意，不管是正常中断结束方式，还是特殊中断结束方式，在一个中断服务子程序结束时，对于级联使用的8259A都必须发两次中断结束命令，一次是发给主片的，另一次则是发给从片的。

3. 屏蔽中断源的方式

8259A的8个中断请求都可根据需要单独屏蔽，屏蔽是通过编程使得屏蔽寄存器IMR相应位置0或置1，从而允许或禁止该位所对应的中断。8259A有两种屏蔽方式。

1）普通屏蔽方式

在普通屏蔽方式中，将IMR某位置1，则它对应的IR_i就被屏蔽，从而使这个中断请求不能从8259A送到CPU。如果该位置0，则允许该IR_i中断传送给CPU。

2）特殊屏蔽方式

在有些情况下，希望一个中断服务程序能动态地改变系统的优先权结构。例如，在执行一个中断服务程序时，可能希望优先级别比正在服务的中断源低的中断能够中断当前的中断服务程序。但在全嵌套方式中，8259A会禁止所有比当前中断服务程序优先级别低的IR_i产生中断。所以，只要当前服务中断的ISR位未被复位，较低级的中断请求在发出EOI命令之前仍不会得到响应。

为解决这个问题，8259A提供了一种特殊屏蔽方式(Special Mask Mode，SMM)。其原理是，在IR_i的处理中，若希望使除IR_i以外的所有IR中断请求均可被响应，则首先设置特殊屏蔽方式，再编程将IR_i屏蔽掉(使IMR中的IM_i位置1)，这样就会使ISR的IS_i位复位。这时，除了正在服务的这级中断被屏蔽(不允许产生进一步中断)外，其他各级中断全部被开放。

特殊中断屏蔽方式提供了允许较低优先级中断源得到响应的特殊手段。但在这种方式下，由于它打乱了正常的全嵌套结构，被处理的程序不见得是当前优先级最高的事件，所以不能用正常EOI命令来使其ISR位复位。但在退出SMM方式之后，仍可用正常EOI命令来结束中断服务。

4. 中断触发方式

外设的中断请求信号从8259A的引脚IR引入，根据实际需要，8259A IR引脚的中断触发方式可分成如下两种。

1）边沿触发方式

8259A的引脚IR_n上出现上升沿表示有中断请求，高电平并不表示有中断请求。

2）电平触发方式

8259A 的引脚 IR_n 上出现高电平表示有中断请求。这种方式下，应注意及时撤除高电平，否则可能引起不应该有的第二次中断。

无论是边沿触发还是电平触发，中断请求信号 IR 都应维持足够的宽度，即在第一个中断响应信号$\overline{INTA}$结束之前 IR 都必须保持高电平。如果 IR 信号提前变为低电平，8259A 就会自动假设这个中断请求来自引脚 IR_7。这种办法能够有效地防止由 IR 输入端上严重的噪声尖峰而产生的中断。为实现这一点，对应 IR_7 的中断服务子程序可只执行一条返回指令，从而滤除这种中断。但如果 IR_7 另有他用，仍可通过读 ISR 状态来识别非正常的 IR_7 中断。因为正常的 IR_7 中断会使 ISR 的 IS_7 位置位，而非正常的 IR_7 中断不会使 ISR 的 IS_7 位置位。

5. 级联工作方式

当中断源超过 8 个，就无法用一片 8259A 来进行管理，这时可采用 8259A 的级联工作方式。指定一片 8259A 为主控芯片（主片），它的 INT 接到 CPU 上。而其余的 8259A 芯片均作为从属芯片（从片），其 INT 输出分别接到主控芯片的 IR 输入端。由于 8259A 有 8 个 IR 输入端，故一个主控 8259A 可以连接 8 片从属 8259A，最多允许有 64 个 IR 中断请求输入。

由一片主控 8259A 和两片从属 8259A 构成的级联中断系统如图 6-29 所示。图中 3 个 8259A 均有各自的地址，由$\overline{CS}$和 A_0 来决定。主片 8259A 的 CAS_0～CAS_2 作为输出连接到从片的 CAS_0～CAS_2 上，而两个从片的 INT 分别接主控芯片的 IR_3 和 IR_6。图中省略了$\overline{CS}$译码器。

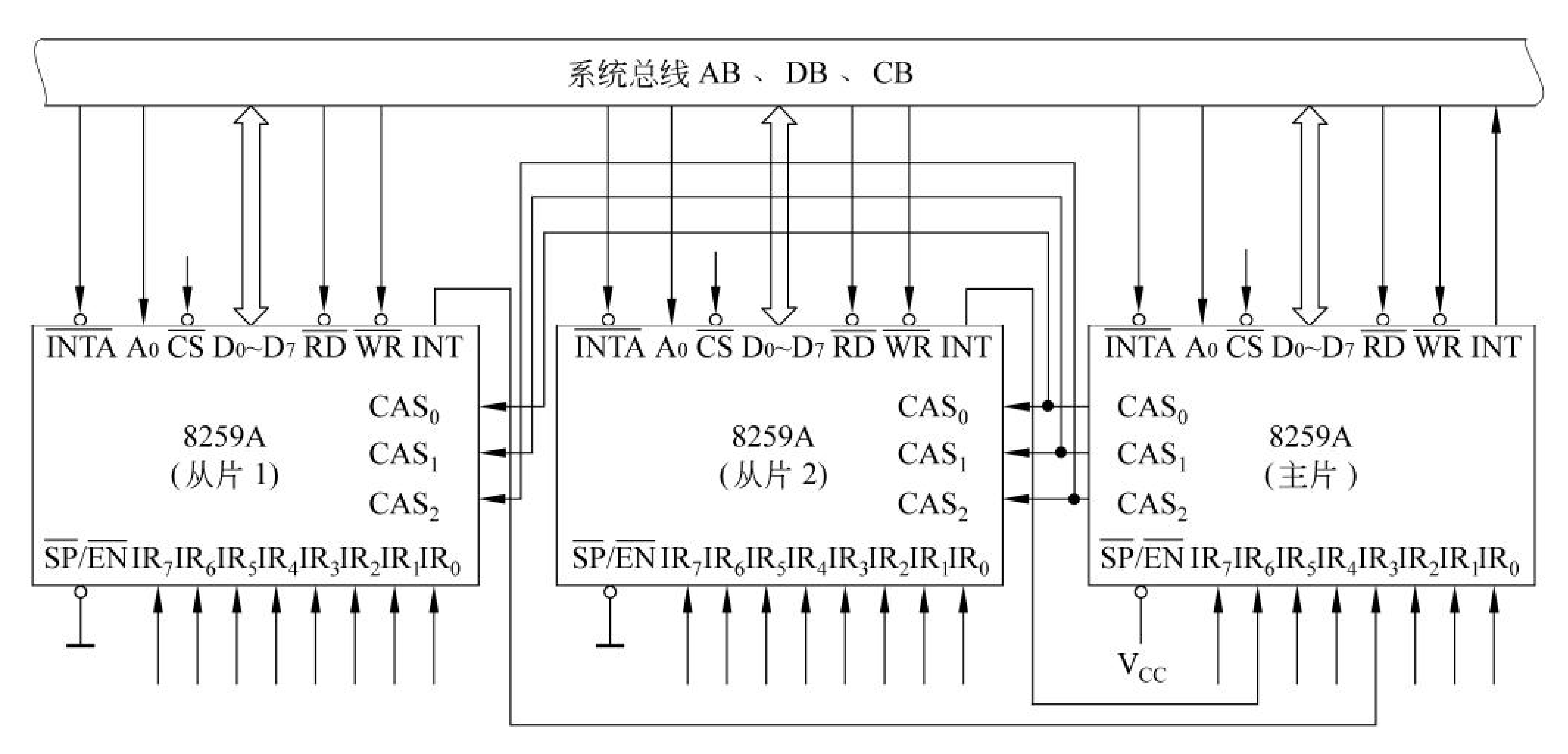


图 6-29　8259A 级联工作方式示意图

在级联系统中，每一片 8259A，不管是主片还是从片，都有各自独立的初始化程序，以便设置各自的工作状态。在中断结束时要连发两次 EOI 命令，分别使主片和相应的从片完成中断结束操作。

在中断响应中，若中断请求是来自从片的IR，则中断响应时主片8259A会通过$CAS_0 \sim CAS_2$来通知相应的从片8259A，而从片8259A即可把IR对应的中断向量码放到数据总线上。

在级联方式下，可采用前面提到的特殊全嵌套方式，以允许从片上优先级更高的IR产生中断。在将主控片初始化为特殊全嵌套方式后，从片的中断响应结束时，要用软件来检查中断状态寄存器ISR的内容，看看本从片上还有无其他中断请求未被处理。如果没有，则连发两个EOI，使从片及主片结束掉中断；若还有其他未被处理的中断，则应只向从片发一个EOI命令，而不向主片发EOI命令。

6.5.4 8259A的初始化编程

8259A是可编程中断控制器，在它工作之前，必须通过软件向其写入控制命令的方法来让它工作在人们所希望的状态下，这就是8259A的编程。控制命令分为初始化命令字ICW（Initialization Command Word）和操作命令字OCW（Operation Command Word），写入8259A后被保存在内部的ICW和OCW寄存器组中。相应地，对8259A的编程也分为初始化编程和操作方式编程两个步骤。

（1）初始化编程：由CPU向8259A送2～4个字节的初始化命令字ICW。在8259A工作之前，必须写入初始化命令字使其处于准备就绪状态。

（2）操作方式编程：由CPU向8259A送3个字节的操作命令字OCW，以规定8259A的操作方式。OCW可在8259A初始化以后的任何时刻写入。

1. 8259A内部寄存器的寻址方法

8259A内部寄存器很多，但靠$\overline{CS}$和A_0将无法满足寻址的需要，因此还要与$\overline{RD}$、$\overline{WR}$和数据线D_4、D_3相配合。表6-1给出了8259A内部寄存器的访问方法。

表6-1 8259A内部寄存器的访问方法

$\overline{CS}$	$\overline{RD}$	$\overline{WR}$	A_0	D_4	D_3	读写操作
0	1	0	0	0	0	写入OCW2
			0	0	1	写入OCW3
			0	1	×	写入ICW1
			1	×	×	写入ICW2、ICW3、ICW4、OCW1(顺序写入)
0	0	1	0	—	—	读出IRR、ISR
			1	—	—	读出IMR

2. 8259A的初始化顺序

从表6-1知，当对8259A进行写时，若I/O地址为奇数($A_0=1$)，则写的对象将包括

4 个寄存器(ICW2、ICW3、ICW4 和 OCW1),即一个 I/O 地址对应了 4 个寄存器。为了区分到底写入的是哪个寄存器,8259A 规定初始化的顺序必须严格按照图 6-30 所规定的顺序依次写入(顺序不可颠倒),即根据顺序来区分不同的寄存器。

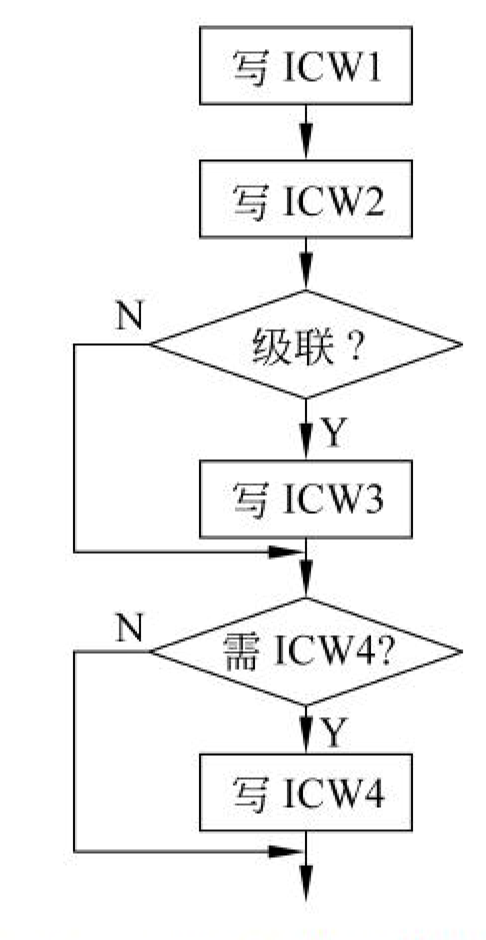


图 6-30　8259A 的初始化顺序

3. 8259A 的内部控制字

8259A 可用于 8080/8085 系统或 8088/8086 系统,用于不同系统时,初始化命令有所不同,以下仅介绍用于 8088/8086 系统时 8259A 的内部控制字。

1) 初始化命令字 ICW

(1) ICW1——初始化字。写 ICW1 的条件：$A_0=0$,$D_4=1$。这时写入的数据被当成 ICW1。写 ICW1 意味着重新初始化 8259A。写 ICW1 的同时,8259A 还做以下几项工作。

① 清除 ISR 和 IMR。

② 将中断优先级设成初始状态：IR_0(最高),IR_7(最低)。

③ 设定为普通屏蔽方式。

④ 采用非自动 EOI 中断结束方式。

⑤ 状态读出电路预置为读 IRR。

ICW1 各位功能如图 6-31 所示(有×符号的位不用,可置为 0)。

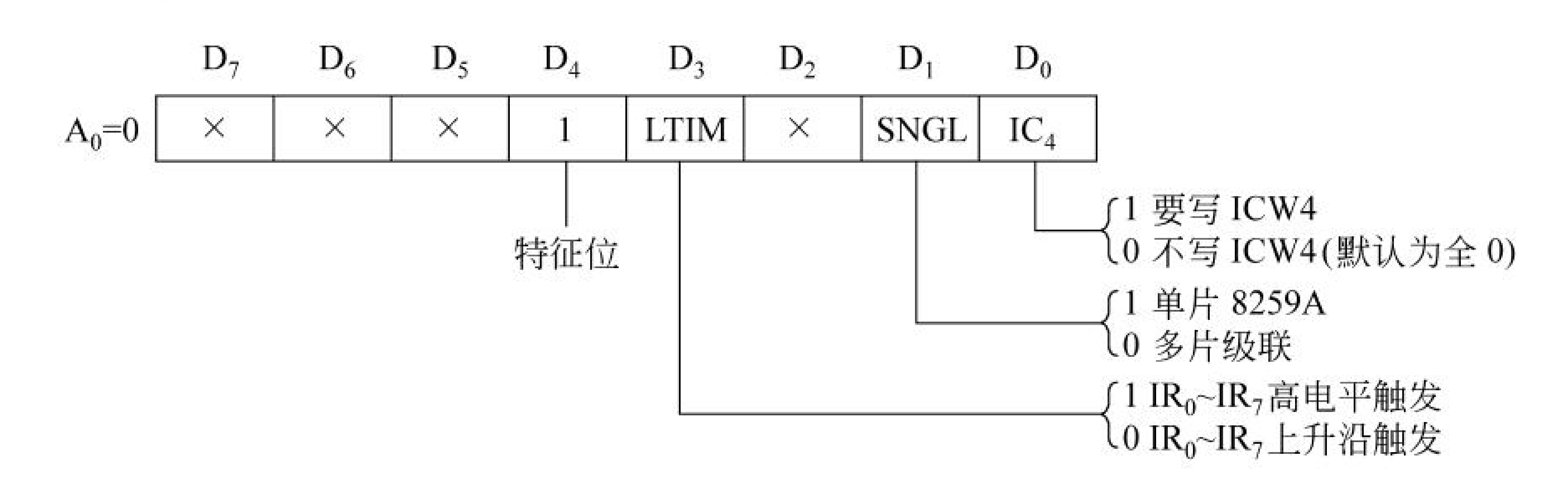


图 6-31　初始化命令字 1(ICW1)

例如：要求上升沿触发、单片 8259、写 ICW4,则 ICW1=00010011B=13H。

(2) ICW2——中断向量码。$A_0=1$ 时,表示要写 ICW2,其格式如图 6-32 所示。ICW2 为中断向量码寄存器,用于存放中断向量(类型)码。CPU 响应中断时,8259A 将该寄存器内容放到数据总线上供 CPU 读取。

初始化时只需确定 $T_6 \sim T_3$。而最低 3 位可以任意(可置为 0),它们最终由 8259A 在中断响应时根据中断源的序号自动填入。

例如：IBM PC 中 ICW2 被初始化为 08H,即 IR_0 的中断向量码为 08H,IR_7 的中断向量码为 0FH 等。

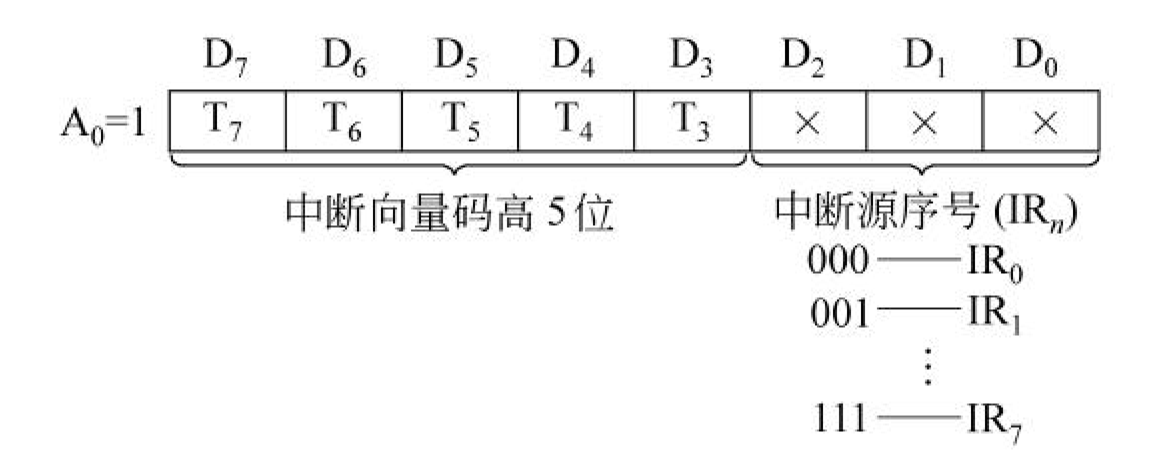


图 6-32　初始化命令字 2(ICW2)

(3) ICW3——级联控制字。ICW3 仅在多片 8259 级联时需要写入。主片 8259A 的 ICW3 与从片的 ICW3 在格式上不同。ICW3 应紧接着 ICW2 写入同一 I/O 地址中。其格式如图 6-33 所示。

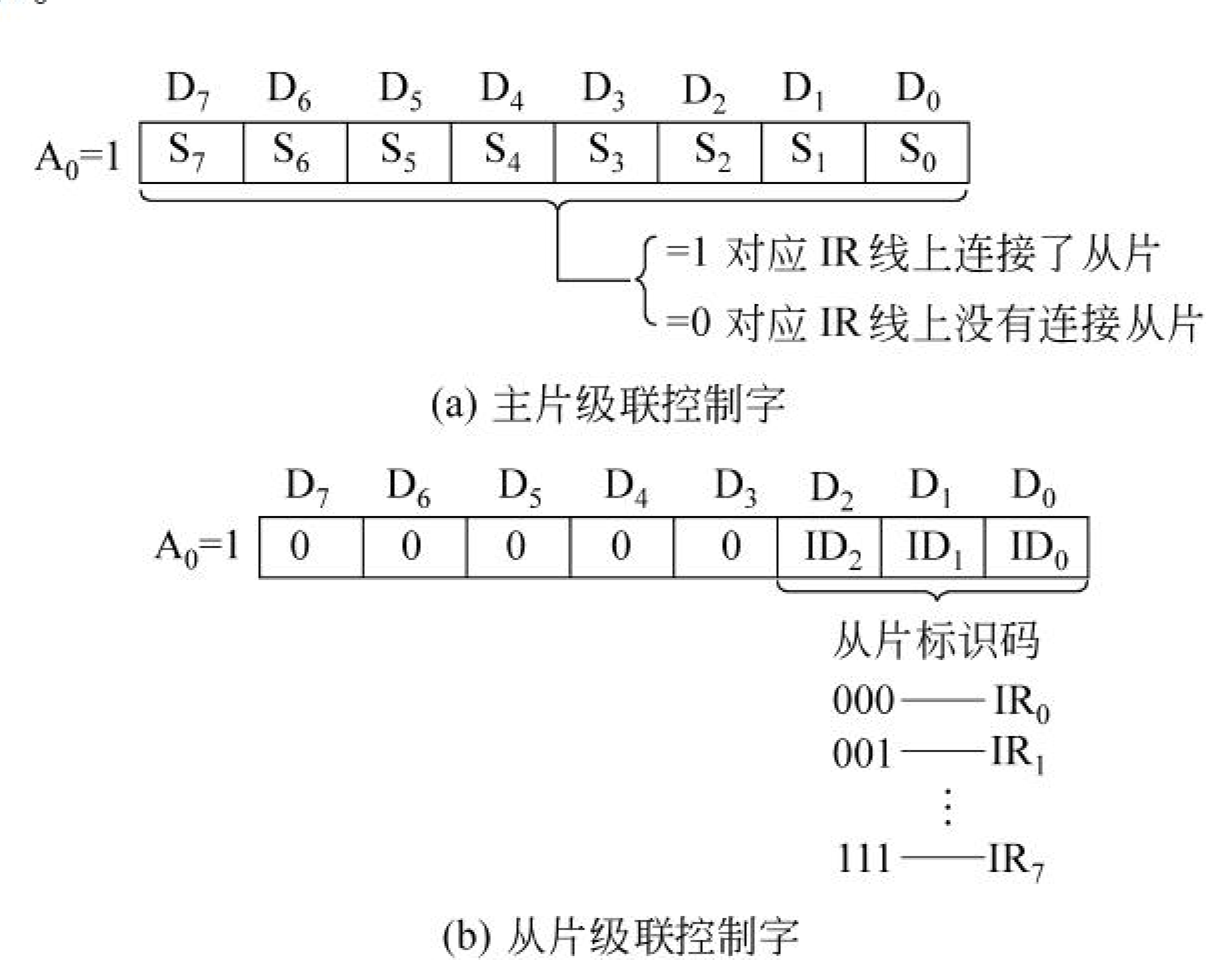


图 6-33　初始化命令字 3(ICW3)

注意，主片 ICW3 各位的设置必须与本主片与从片相连之 IR 线的序号一致。例如，主片的 IR_4 与从片的 INT 连接，则主片 ICW3 的 S_4 位应为 1。

同理，从片标识码也必须与本从片所连接之主片 IR 线的序号一致。例如，某从片的 INT 线与主片的 IR_4 连接，则该从片的 ICW3＝04H。

(4) ICW4——中断结束方式字。ICW4 应紧跟在 ICW3 之后写入同一 I/O 地址中。ICW4 的格式如图 6-34 所示。

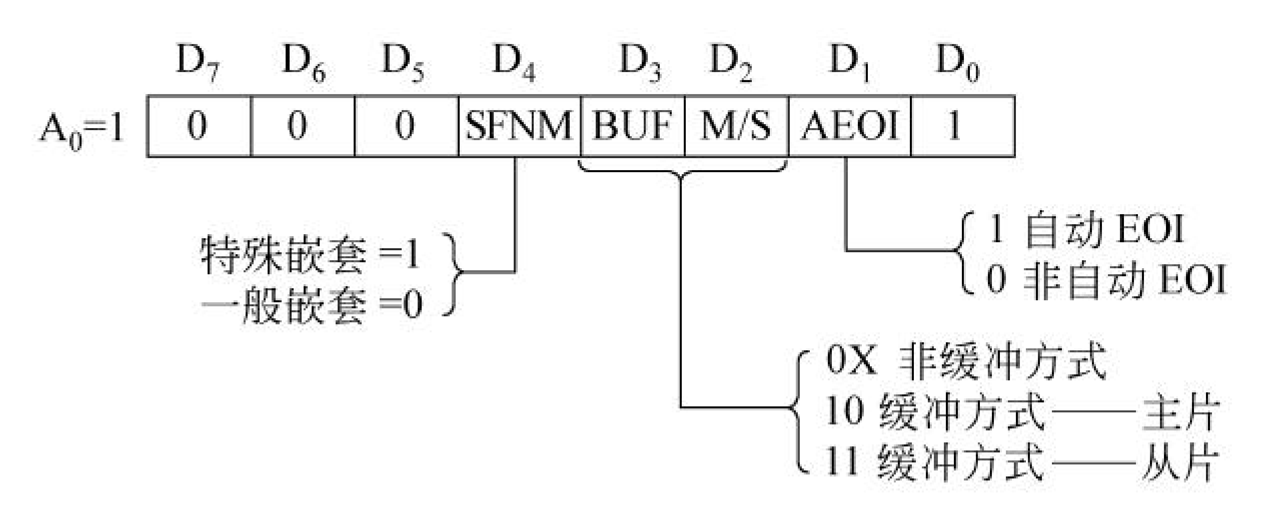


图 6-34　初始化命令字 4(ICW4)

图中的缓冲方式是指 8259A 工作于级联方式时，其数据线与系统总线之间增加一个缓冲器，以增大驱动能力。这时 8259A 把 $\overline{SP}/\overline{EN}$ 作为输出端，输出一个允许信号，用来控制缓冲器的打开与关闭。而主片与从片只能用 D_2（M/S 位）来区分（主片＝0，从片＝1）。在非缓冲方式时，若 8259A 工作在级联方式，$\overline{SP}/\overline{EN}$ 引脚为输入端，用来区分主片（高电平）和从片（低电平）。

2）操作命令字 OCW

操作命令字可用来改变 8259A 的中断控制方式、屏蔽某几个中断源以及读出 8259A 的工作状态信息（IRR、ISR、IMR）。操作命令字在初始化完成后任意时刻均可写入，写的顺序也没有严格要求。但它们对应的端口地址有严格规定，OCW1 必须写入奇地址端口（A_0＝1），OCW2 和 OCW3 必须写入偶地址端口（A_0＝0）。

（1）OCW1——中断屏蔽字。OCW1 用于决定中断请求线 IR_i 被屏蔽否。初始时为全 0（全部允许中断），写入时要求地址线 A_0＝1。OCW1 的格式如图 6-35 所示。

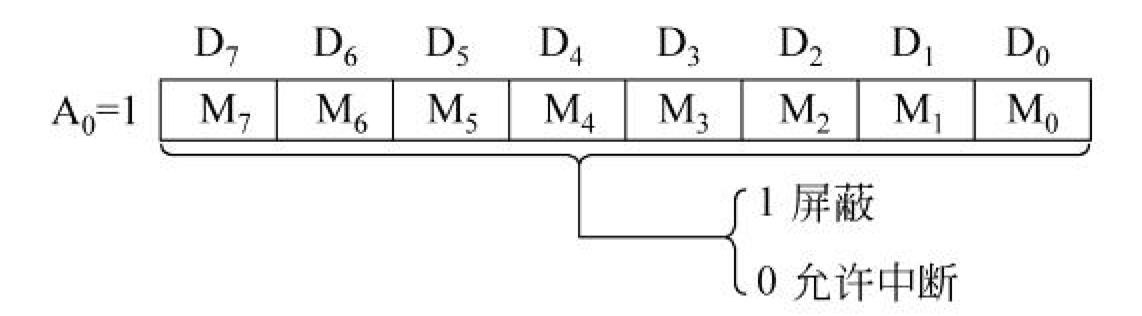


图 6-35　操作命令字 1（OCW1）

（2）OCW2——中断结束和优先级循环。OCW2 的作用是对 8259A 发出中断结束命令 EOI，它还可以控制中断优先级的循环。OCW2 的格式如图 6-36 所示。它与 OCW3 共用一个端口地址，但其特征位 D_4D_3＝00，因此不会发生混淆。OCW2 写入时要求地址线 A_0＝0。

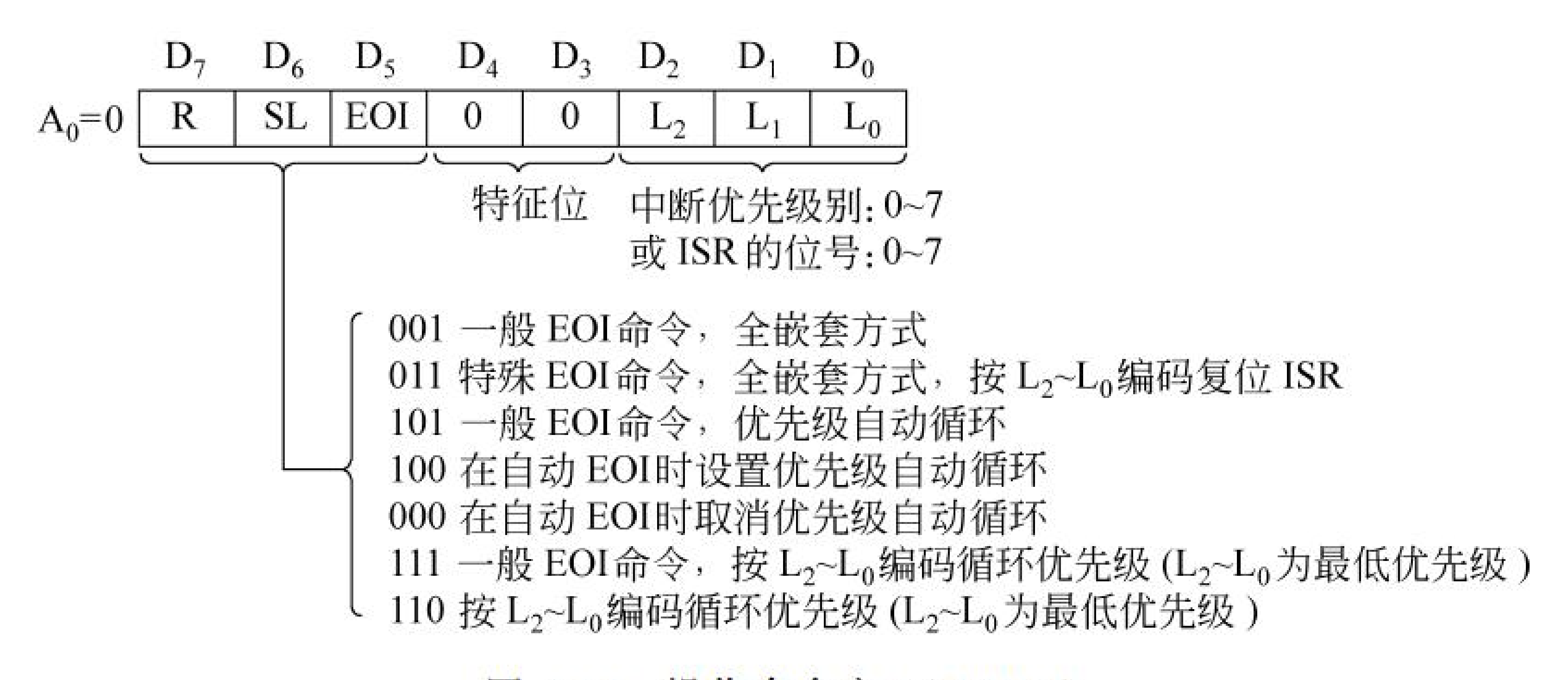


图 6-36　操作命令字 2（OCW2）

① R：优先级循环控制位。R＝0 时表示使用固定优先级，IR_7 最低，IR_0 最高；当 R＝1 时，表示使用循环优先级，一个优先级别的中断服务结束后，它的优先级就变为最低级，而下一个优先级变为最高级。

② SL：特殊循环控制。当 SL＝1 时，使 $L_2 \sim L_0$ 对应的 IR_i 为最低优先级；SL＝0 时，$L_2 \sim L_0$ 的编码无效。

③ EOI：中断结束命令。该位为 1 时，则复位现行中断的 ISR 中的相应位，以便允许

8259A 再为其他中断源服务。在 ICW4 的 AEOI＝0（非自动 EOI）的情况下，需要用 OCW2 来复位现行中断的 ISR 中的相应位。

④ $L_2 \sim L_0$：第一个作用是设定哪个 IR_i 优先级最低，用来改变 8259A 复位后所设置的默认优先权级别；第二个作用是在特殊中断结束命令中指明 ISR 哪一位要被复位。

R、SL、EOI 三者组合所代表的含义见图 6-36 中的说明。

（3）OCW3——屏蔽方式和状态读出控制字。OCW3 的格式如图 6-37 所示，它有以下 3 个功能：

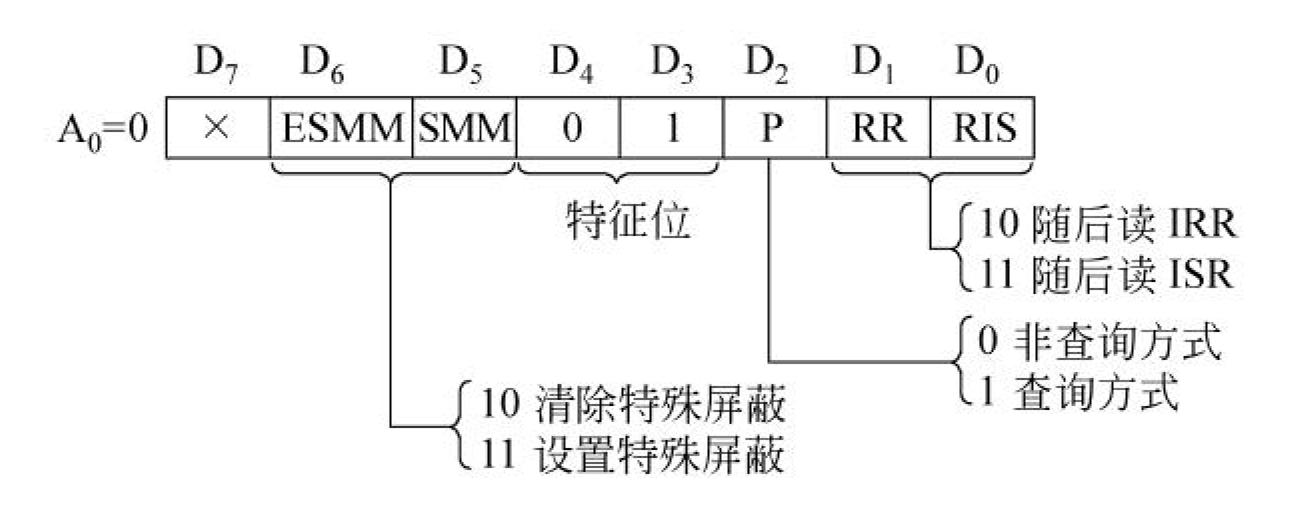


图 6-37　操作命令字 3（OCW3）

① 设置中断屏蔽方式，见图 6-37 中的说明。

② 查询中断请求。当 CPU 禁止中断或不希望 8259A 向 CPU 申请中断时，就可以采用 8259A 的查询工作方式。CPU 先写一个 P＝1 的 OCW3 到 8259A，再对同一地址读入，即可得到图 6-38 所示格式的状态字节。



I	×	×	×	×	R_2	R_1	R_0

图 6-38　8259A 中断状态查询结果

若 I＝1，则表示本片 8259A 的 $IR_0 \sim IR_7$ 中有中断请求产生，其中最高优先级的 IR 线的编码由 $R_2 \sim R_0$ 给出；I＝0 表示无中断请求产生。（此查询步骤可反复执行，以响应多个同时发生的中断。）

③ 读 8259A 状态。可用 OCW3 命令控制读出 IRR、ISR 和 IMR 的内容。CPU 先写一个 RR RIS＝10 的 OCW3 到 8259A，再对同一地址读，即可读入 IRR 的内容；CPU 先写一个 RR RIS＝11 的 OCW3 到 8259A，再对同一地址读，即可读入 ISR 的内容。而当 A_0＝1（奇地址）时读 8259A，则读出的都是 IMR 的内容（不依赖于 OCW3）。

4. 8259A 编程举例

下面以 IBM PC/AT（80286）微机中的 8259A 为例说明其编程方法。

在 286 以上的 PC 中，共使用了两片 8259A（新型的 PC 中已将中断控制器集成到了芯片组中，但功能上与 8259A 完全兼容），两片级联使用，共可管理 15 级中断。各级中断的用途如表 6-2 所示。

主片 8259A 的 IRQ_2（即 IR_2）中断请求端用于级联从片 8259A，所以相当于主片的 IRQ_2 又扩展了 8 个中断请求端 $IRQ_8 \sim IRQ_{15}$。

表 6-2 IBM PC/AT 的中断源和类型号

中断向量地址指针	8259A 引脚	中断类型号	优先级	中断源
00020H	主片 IR_0	08H	0(最高)	定时器
00024H	主片 IR_1	09H	1	键盘
00028H	主片 IR_2	0AH	2	从片 8259A
001C0H	从片 IR_0	70H	3	时钟/日历
001C4H	从片 IR_1	71H	4	IRQ_9(保留)
001C8H	从片 IR_2	72H	5	IRQ_{10}(保留)
001CCH	从片 IR_3	73H	6	IRQ_{11}(保留)
001D0H	从片 IR_4	74H	7	IRQ_{12}(保留)
001D4H	从片 IR_5	75H	8	协处理器
001D8H	从片 IR_6	76H	9	硬盘控制器
001DCH	从片 IR_7	77H	10	IRQ_{15}(保留)
0002CH	主片 IR_3	0BH	11	异步通信口(COM2)
00030H	主片 IR_4	0CH	12	异步通信口(COM1)
00034H	主片 IR_5	0DH	13	并行打印口 2
00038H	主片 IR_6	0EH	14	软盘驱动器
0003CH	主片 IR_7	0FH	15(最低)	并行打印口 1

主片 8259A 的端口地址为 20H、21H,中断类型码为 08H～0FH;从片 8259A 的端口地址为 A0H、A1H,中断类型码为 70H～77H。主片的 8 级中断已全被系统使用(其中 IRQ_2 被从片占用),从片尚保留 4 级未用。其中 IRQ_0 用于日历时钟中断(08H),IRQ_1 用于键盘中断(09H)。扩展的 IRQ_8 用于实时时钟中断,IRQ_{13} 来自协处理器 80287。除上述中断请求信号外,所有其他的中断请求信号都来自 I/O 通道的扩展板。

1) 8259A 初始化编程

```
;主片 8259A 的初始化
        MOV AL,11H              ;写入 ICW1,设定边沿触发,级联方式
        OUT 20H,AL
        JMP INTR1               ;延时,等待 8259A 操作结束,下同
INTR1:  MOV AL,08H              ;写入 ICW2,设定 IRQ0 的中断类型码为 08H
        OUT 21H,AL
        JMP INTR2
INTR2:  MOV AL,04H              ;写入 ICW3,设定主片 IRQ2 级联从片
        OUT 21H,AL
        JMP INTR3
INTR3:  MOV AL,11H              ;写入 ICW4,设定特殊全嵌套方式,一般 EOI 方式
        OUT 21H,AL
        …
  ;从片 8259A 的初始化
        MOV AL,11H              ;写入 ICW1,设定边沿触发,级联方式
        OUT 0A0H,AL
        JMP INTR5
```

```
INTR5:  MOV AL,70H          ;写入 ICW2,设定从片 IR0,即 IRQ8的中断类型码为 70H
        OUT 0A1H,AL
        JMP INTR6
INTR6:  MOV AL,02H          ;写入 ICW3,设定从片级联到主片的 IRQ2
        OUT 0A1H,AL
        JMP INTR7
INTR7:  MOV AL,01H          ;写入 ICW4,设定普通全嵌套方式,一般 EOI 方式
        OUT 0A1H,AL
        …
```

2）级联工作编程

当来自某个从片的中断请求进入服务时，主片的优先权控制逻辑不封锁这个从片，从而使来自从片的更高优先级的中断请求能被主片所识别，并向 CPU 发出中断请求信号。因此，中断服务子程序结束时必须用软件来检查被服务的中断是否是该从片中唯一的中断请求。先向从片发出一个 EOI 命令，清除已完成服务的 ISR 位。然后再读出 ISR 的内容，检查它是否为 0。如果 ISR 的内容为 0，则向主片发一个 EOI 命令，清除与从片相对应的 ISR 位；否则，就不向主片发 EOI 命令，继续进行从片的中断处理，直到 ISR 的内容为 0，再向主片发出 EOI 命令。程序段如下：

```
;读 ISR的内容
      MOV AL,0BH            ;写入 OCW3,读 ISR命令
      OUT 0A0H,AL
      NOP                   ;延时,等待 8259A操作结束
      IN AL,0A0H            ;读出 ISR
      …
;向从片发 EOI命令
      MOV AL,20H
      OUT 0A0H,AL           ;写从片 EOI命令
      …
;向主片发 EOI命令
      MOV AL,20H
      OUT 20H,AL            ;写主片 EOI命令
      …
```

6.5.5 中断程序设计概述

在 PC 中，8259A 的初始化已由操作系统完成，用户不需要再对 8259A 进行初始化。一般情况下，用户向 8259A 写的控制字只有 EOI 命令，偶尔可能也要重写中断屏蔽字(但程序运行结束后，应恢复原值)。用户在编制中断程序时主要应注意 4 个方面的问题：中断服务子程序格式、保护原中断向量、设置自己的中断向量、恢复原中断向量。中断程序设计的一般过程(PC 中主片 8259A 的 I/O 地址为 20H 和 21H)如下。

(1) 确定要使用的中断类型号。中断类型号不能随便用，有些中断类型号已被系统

所占用，若强行使用可能会使系统崩溃。可供用户使用的中断类型号为 60H～66H 和 68H～6FH。

(2) 保存原中断向量。在将自己的中断服务子程序的入口地址设置到中断向量表中之前，应先保存该地址中原来的内容，这可用 INT 21H 中的 35H 号功能完成。取出的中断向量被放在 ES:BX 中，ES 为段地址，BX 为偏移地址。取出的中断向量可保存在用户程序的附加段或数据段中，以便退出前恢复。

(3) 设置自己的中断向量。将自己编写的中断服务子程序的首地址存入中断向量表的相应表项中，可以用 DOS 功能调用的 25H 号功能完成。在调用 25H 号功能前，中断服务子程序所在段的段地址应放在 DS 中，中断服务子程序的偏移地址放在 DX 中。

(4) 设置中断屏蔽字(可选)。若编写的是硬件中断程序，应将所使用的硬件中断对应的 8259A 的中断屏蔽位开放。方法参考前面有关 8259A 的寄存器设置方法和初始化程序。

(5) CPU 开中断。前面的工作完成后，就可打开 CPU 的中断标志位，以便让 CPU 响应中断。

(6) 恢复原中断向量。程序退出前一定要恢复原中断向量。这是因为你的程序一旦退出，该存储区内容将不可预料，若又产生同类型中断，CPU 将转移到这个不可预料的内存区去执行，其后果很可能是系统崩溃、死机。

另外，在编写中断服务子程序时，要使 CPU 在中断服务子程序中停留的时间越短越好，这就要求中断服务子程序要编写得短小精悍，能放在主程序中完成的任务就不要由中断服务子程序来完成。

下面给出中断服务子程序及其主程序的典型形式。

(1) PC 中中断服务程序的一般形式(下划线处为特别要注意的地方)：

```
MY_INT PROC FAR
PUSH<需要保护的寄存器 1>
PUSH<需要保护的寄存器 2>
…
PUSH<需要保护的寄存器 i>
STI
…
<中断服务程序主体>
…
CLI
POP<在入口处保护的寄存器 i>
…
POP<在入口处保护的寄存器 2>
POP<在入口处保护的寄存器 1>
MOV AL,20H                    ;EOI 命令,00100000B
OUT 20H,AL                    ;写 OCW2
IRET
MY_INT ENDP
```

(2) 主程序形式：

```
    …
;保护原中断向量表内容
    MOV AH,35H
    MOV AL,<中断类型码>          ;将要保护的中断源的中断类型码送 AL
    INT 21H                      ;取原中断向量(放在 ES:BX中)
    MOV SAVE_IP,BX               ;把取回的中断向量保存在本程序的
    MOV SAVE_CS,ES               ;数据段中

;设置自己的中断服务程序入口
    PUSH DS
    MOV DX,OFFSET MY_INT
    MOV AX,SEG MY_INT
    MOV DS,AX                    ;DS:DX的内容为中断服务程序的首地址
    MOV AH,25H
    MOV AL,<中断类型码>          ;将自己的中断类型码送 AL
    INT 21H                      ;设新中断向量
    POP DS
    STI                          ;开中断
    …
    <主程序放在这里>
    …
;退出程序前恢复原中断向量内容
    CLI
    PUSH DS
    MOV DX,SAVE_IP
    MOV AX,SAVE_CS
    MOV DS,AX
    MOV AH,25H
    MOV AL,<中断类型码>          ;将原中断类型码送 AL
    INT 21H
    POP DS
    STI
    <退出主程序,返回 DOS>
    …
```

有关中断程序设计的进一步的详细描述参阅相关资料和书籍。

习　　题

6.1　输入输出系统主要由哪几个部分组成？主要有哪些特点？

6.2　I/O 接口的主要功能有哪些？有哪两种编址方式？在 8088/8086 系统中采用哪一

种编址方式？

6.3 试比较4种基本输入输出方法的特点。

6.4 主机与外部设备进行数据传送时，采用哪一种传送方式CPU的效率最高？

6.5 某输入接口的地址为0E54H，输出接口的地址为01FBH，分别利用74LS244和74LS273作为输入和输出接口。画出其与8088系统总线的连接图；并编写程序，使当输入接口的D_1、D_4和D_7位同时为1时，CPU将内存中DATA为首址的20个单元的数据从输出接口输出，若不满足上述条件则等待。

6.6 为什么74LS244只能作为输入接口？74LS273只能作为输出接口？

6.7 利用74LS244作为输入接口（端口地址为01F2H）连接8个开关K_0～K_7，用74LS273作为输出接口（端口地址为01F3H）连接8个发光二极管。

(1) 画出芯片与8088系统总线的连接图，并利用74LS138设计地址译码电路。

(2) 编写实现下述功能的程序段。

① 若8个开关K_7～K_0全部闭合，则使8个发光二极管亮。

② 若开关高4位（K_4～K_7）全部闭合，则使连接到74LS273高4位的发光管亮。

③ 若开关低4位（K_3～K_0）闭合，则使连接到74LS273低4位的发光管亮。

④ 其他情况不做任何处理。

6.8 8088/8086系统如何确定硬件中断服务程序的入口地址？

6.9 中断向量表的作用是什么？如何设置中断向量表？

6.10 INTR中断和NMI中断有什么区别？

6.11 试说明8088 CPU可屏蔽中断的处理过程。

6.12 CPU满足什么条件能够响应可屏蔽中断？

6.13 8259A有哪几种优先级控制方式？一个外中断服务程序的第一条指令通常为STI，其目的是什么？

6.14 试编写8259A的初始化程序：系统中仅有一片8259A，允许8个中断源边沿触发，不需要缓冲，一般全嵌套方式工作，中断向量为40H。

6.15 单片8259A能够管理多少级可屏蔽中断？若用3片级联能管理多少级可屏蔽中断？

6.16 具备何种条件能够作为输入接口？具备何种条件能够作为输出接口？

6.17 已知SP＝0100H，SS＝3500H，CS＝9000H，IP＝0200H，[00020H]＝7FH，[00021H]＝1AH，[00022H]＝07H，[00023H]＝6CH，在地址为90200 H开始的连续两个单元中存放着一条两字节指令INT 8。试指出在执行该指令并进入相应的中断子程序时，SP、SS、IP、CS寄存器的内容以及SP所指向的字单元的内容。



第7章 常用数字接口电路

引言：

CPU与外部设备之间的信息交换是通过接口电路来实现的，它通过接口接收外部设备送出的信息，又将信息发送给外设。接口成为这种信息交换的必经通道，起着一种“桥梁”的作用。没有接口电路，计算机也就无法与外部设备进行通信。

利用三态门、锁存器这样的简单接口可以实现一个简易“家庭安全防盗系统”的设计，但简单接口芯片在功能上比较单一，使用中具有较大的局限性，只适合于简单外部设备的连接。本章将介绍几种常用的可编程I/O数字接口芯片。

教学目的：

(1) 了解并行通信及串行通信的一般概念；

(2) 掌握几种可编程接口芯片的应用。

接口是输入输出系统中一个重要的组成部分，处理器与外部设备之间的信息交换需要通过接口实现。接口所担当的这种“角色”决定了它需要完成信息缓冲、信息变换、电平转换、数据存取和传送以及联络控制等工作，这些工作分别由接口电路的两大部分——和计算机连接的总线接口以及与外部设备连接的外设接口来实现。总线接口一般包括内部寄存器、存取逻辑和传送控制逻辑电路等，主要负责数据缓冲、传输管理等工作；而外设接口则负责与外部设备通信时的联络和控制以及电平和信息变换等。本章所讨论的接口电路都是指外设接口。

接口电路从总的功能上可以分为输入接口和输出接口，分别完成信息的输入和输出；从传送方式上又可分为并行接口和串行接口；另外，从所传送信息的类型上还可分为数字量的输入输出接口及模拟量的输入输出接口；本章主要介绍用于数字信息传送的典型的可编程I/O接口芯片。

一般来讲，接口芯片的内部都包括两部分，一部分负责和计算机系统总线的连接，另一部分负责和外部设备的连接，其连接示意图如图6-2所示。负责与系统总线连接的部分主要包括：数据信号线、控制信号线和地址信号线。数据信号线除实现数据的接收和发送外，还负责传送CPU发给接口的编程命令及接口送出的状态信息；控制信号线主要是读/写控制信号，由于多数系统中对外设的读写和存储器的读写是相互独立的，因此接口的读写信号$\overline{RD}$和$\overline{WR}$应分别与系统读写外设的信号$\overline{IOR}$和$\overline{IOW}$相连；地址信号线一般通过译码电路连接到接口的片选端，从而确定接口所占的地址或地址范围。

近年来，随着超大规模集成电路技术的发展，已有各种通用和专用的接口芯片问世，

为微型机的应用打下了良好的硬件基础。第6章中介绍了一些简单的接口电路芯片及其应用。这些芯片一般只适合于慢速且功能比较简单的外设，难以满足各种应用控制系统的要求。本章将在7.2节和7.3节中介绍两种可编程接口芯片的工作原理和应用方法。

7.1 并行通信与串行通信

计算机与计算机之间或计算机与外部设备之间的信息交换称为通信。计算机的通信有两种基本方式：并行通信和串行通信。在通信过程中，如果能够同时传送数据的所有位(位数由机器的字长决定)，就称为并行通信；如果数据是逐位顺序传送，则称为串行通信。计算机与外设间的接口按照通信方式的不同，相应地分为并行接口和串行接口。并行通信和串行通信是指接口与外部设备一侧的通信方式，与CPU之间的通信都是并行的。

7.1.1 并行通信

1. 并行接口的特点

由于多数的I/O设备，特别是系统基本I/O设备都采用并行数据传送，因此并行接口的应用十分普遍。一般来讲，并行接口具有以下主要特点。

(1) 以数据字节或字为单位进行数据传送，两个功能模块间有多位数据同时进行数据传送，速度快、效率高。

(2) 适合近距离传送。由于并行通信所需要的数据线路较多，造价高，且易产生干扰。因此并行通信通常都用于近距离、高速数据交换的场合。

(3) 并行传送方式中，8位、16位或4字节的数据是同时传输的，因此在并行接口与外部设备进行数据交换时，即使只需要传送一位，也是一次输入输出8位、16位或4字节。

(4) 串行传送的信息有固定格式要求，并行传送的信息不要求固定格式。

2. 并行接口的类型

并行接口从不同的角度可以有以下几种分类方法。

(1) 从数据传送的方向上分，可以分为输入接口和输出接口。用于将信息从外部设备输入到系统的接口称为输入接口；反之，将信息从系统送入到外部设备的接口称为输出接口。对输入和输出接口的基本要求在第6章中已讲到，即输入接口必须具有对数据的控制能力，而输出接口必须具有对数据的锁存能力。

(2) 从传输数据的形式上分，可以分为单向传送接口和双向传送接口。单向传送接口的传送方向是确定的，即在系统中只能作为输入接口或者输出接口；而双向传送接口则既可以作为输入接口，也可以作为输出接口。

(3) 从接口的电路结构上分,可以分为简单接口(或硬接线接口)和可编程接口。简单接口的工作方式和功能比较单一,只能进行数据的传送,不能产生系统需要的各种控制和状态信息。如第 6 章中介绍的三态门接口和锁存器接口就是典型的简单接口电路。这类接口电路主要用于连接不需任何联络信号就可实现并行数据传送的简单、低速的外部设备。

可编程接口电路能够通过软件编程的方法改变接口的工作方式及功能,具有较好的适应性和灵活性,在微机系统中得到了广泛的应用。这类芯片的工作原理将在 7.2 节和 7.3 节中介绍。

(4) 从传送信息的类型上分,接口电路又可分为数字接口和模拟接口。在本章和第 6 章中所介绍的接口电路都是用于传输数字信息的数字接口,本书第 8 章将介绍用于进行模拟量传送的模拟接口。

7.1.2 串行通信

串行通信是指两个功能模块只通过一条或两条数据线进行数据交换。发送方将数据分解为二进制位,一位接一位地顺序通过单条数据线发送,接收方则一位一位地从单条数据线上接收,并将其重新组装成一个数据。串行通信数据线路少,造价低,适合于远距离传送。但由于数据是一位一位传送的,故速度较慢。

1. 串行数据传送方式

串行通信是一位一位通过同一信号线进行数据传送的方式。按照数据流的方向可分为 3 种基本传送方式:全双工、半双工和单工。

如果串行通信的通路只有一条,此时发送信息和接收信息就不能同时进行,只能采用分时使用线路的方法,如在 A 发送信息时,B 只能接收;而当 B 发送信息时,则 A 只能接收。这种串行通信的工作方式称为半双工通信方式,如图 7-1(a)所示。

如果有两条通路,则发送信息和接收信息就可以同时进行。如当 A 发送信息、B 接收时,B 也能够同时利用另一条通路发送信息而由 A 接收。这种工作方式称为全双工通信方式,如图 7-1(b)所示。

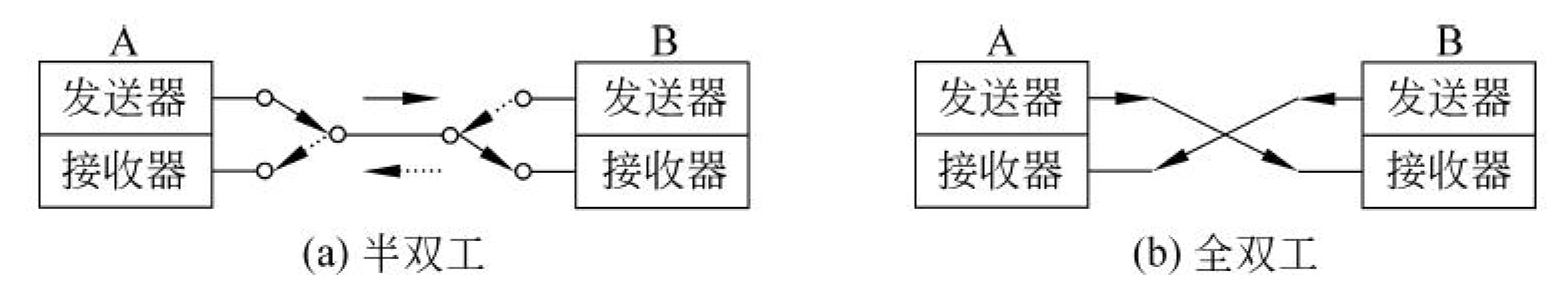


图 7-1 串行通信工作方式

除了半双工和全双工通信外,还有一种单工通信方式,它只允许一个方向传送信息,而不允许反向传输。这种方式在实际应用中较少见。

2. 调制与解调

计算机通信时发送接收的信息均是数字信号,其占用的频带很宽,约为几 MHz 甚至

更高;但目前长距离通信时采用的传统电话线路频带很窄,大约仅有 4kHz。直接传送必然会造成信号的严重畸变,大大降低了通信的可靠性。所以在长距离通信时,为了确保数据的正常传送,一般都要在传送前把信号转换成适合于传送的形式,传送到目的地后在再恢复成原始信号。这个转换工作可利用调制解调器(modem)来实现。

在发送站,调制解调器把“1”和“0”的数字脉冲信号调制在载波信号上;承载了数字信息的载波信号在普通电话网络系统中传送;在目的站,调制解调器把承载了数字信息的载波信号再恢复成原来的“1”和“0”数字脉冲信号。

信号的调制方法主要有 3 种:调频、调幅和调相。当调制信号为数字信号时,这 3 种调制方法又分别称为频移键控法(Frequency Shift Keying,FSK)、幅移键控法(Amplitude Shift Keying,ASK)和相移键控法(Phase Shift Keying,PSK)。

(1) 调频就是把数字信号的“1”和“0”调制成不同频率的模拟信号,例如用 1200Hz 的信号表示“0”,用 2400Hz 的信号表示“1”。接收方根据载波信号的频率就可知道传输的信息是“1”还是“0”。

(2) 调幅就是把数字信号的“1”和“0”调制成不同幅度的模拟信号,但频率保持不变。例如载波信号的幅度大于 8V 时表示“0”,载波信号的幅度小于 3V 时表示“1”。

(3) 调相就是把数字信号的“1”和“0”调制成不同相位的模拟信号,但频率和幅度均保持不变。例如载波信号的相位为 0°时表示“0”,载波信号的相位为 180°时表示“1”。

3. 同步通信和异步通信

串行通信的数据是逐位传送的,发送方发送的每一位都具有固定的时间间隔,这就要求接收方也要按照发送方同样的时间间隔来接收每一位。不仅如此,接收方还要确定一个信息组的开始和结束。为此,串行通信对传送数据的格式作了严格的规定。不同的串行通信方式具有不同的数据格式。下面简单介绍一下常用的两种基本串行通信方式——同步通信和异步通信及其数据传送格式。

1) 同步通信

所谓同步通信,是指在约定的通信速率下,发送端和接收端的时钟信号频率和相位始终保持一致(同步)。这就保证了通信双方在发送和接收数据时具有完全一致的定时关系。

同步通信把许多字符组成一个信息组(或称为信息帧),每帧的开始用同步字符来指示。由于发送和接收的双方采用同一时钟,所以在传送数据的同时还要传送时钟信号,以便接收方可以用时钟信号来确定每个信息位。

同步通信要求在传输线路上始终保持连续的字符位流,若计算机没有数据传输,则线路上要用专用的“空闲”字符或同步字符填充。

同步通信传送信息的位数几乎不受限制,通常一次通信传送的数据有几十到几千个字节,通信效率较高。但它要求在通信中保持精确的同步时钟,所以其发送器和接收器比较复杂,成本也较高,一般用于传送速率要求较高的场合。

用于同步通信的数据格式有许多种,图 7-2 表示了常见的几种数据格式。在图 7-2 中,除数据部分的长度可变外,其他均为 8 位。其中图 7-2(a)为单同步格式,传送一帧数

据仅使用一个同步字符。当接收端收到并识别出一个完整同步字符后就连续接收数据。一帧数据结束，进行 CRC 校验。图 7-2(b)为双同步字格式，这时利用两个同步字符进行同步。图 7-2(c)为同步数据链路控制(SDLC)规程所规定的数据格式，而图 7-2(e)称为高级数据链路控制(HDLC)规程所规定的数据格式，它们均用于同步通信。这两种规程的细节本书不做详细说明。图 7-2(d)则是一种外同步方式所采用的数据格式，对这种方式，在发送的一帧数据中不包含同步字符。同步信号 SYNC 通过专门的控制线加到串行接口上。当 SYNC 一到达，表明数据部分开始，接口就连续接收数据和 CRC 校验码。

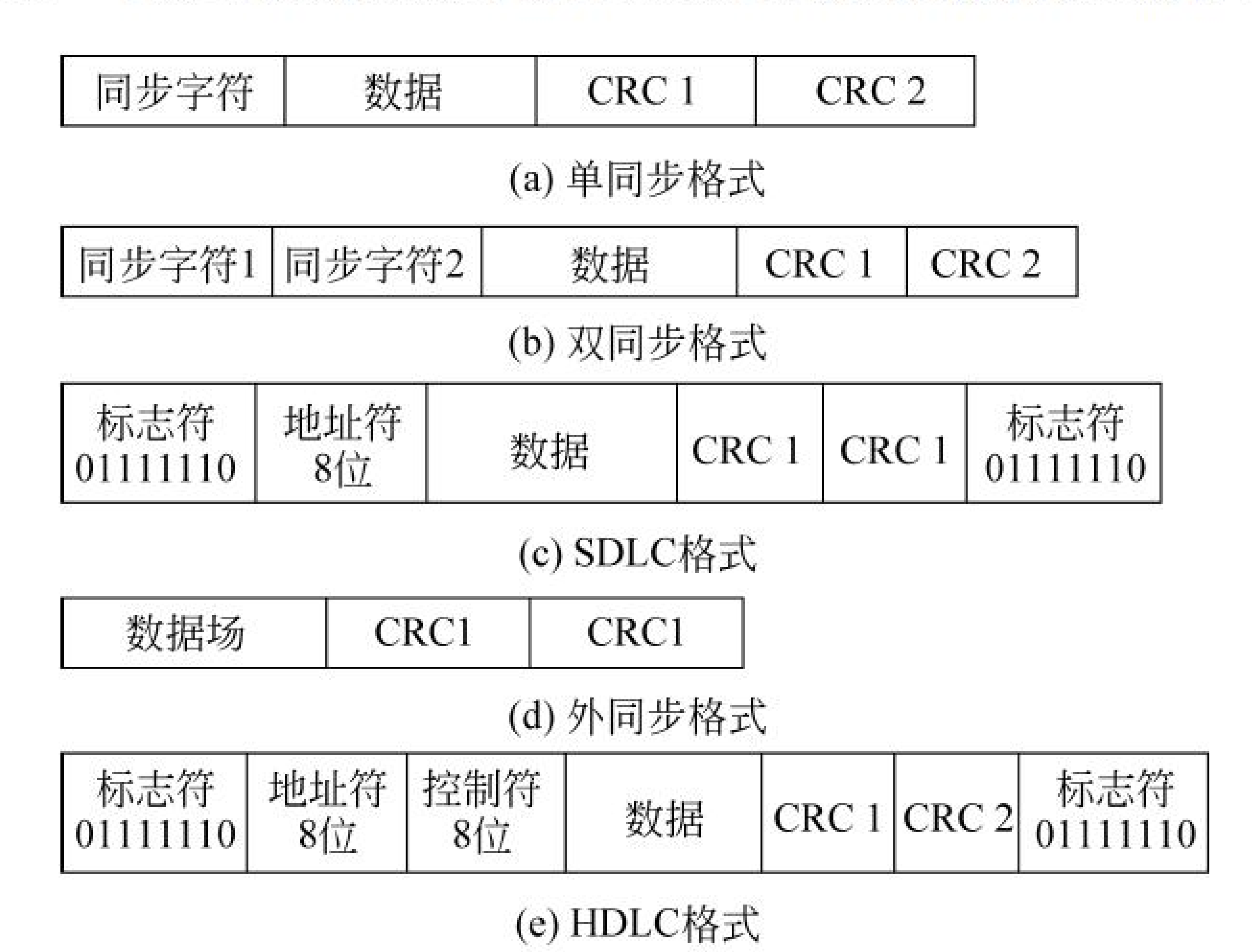


图 7-2　常见的几种同步通信数据格式

CRC(Cyclic Redundancy Checks)的意思是循环冗余校验码。它用于检验在传输过程中是否出现错误，是保证传输可靠性的重要手段之一。

2）异步通信

异步通信是指通信中两个字符之间的时间间隔是不固定的，而在一个字符内各位的时间间隔是固定的。

异步通信规定字符由起始位(Start Bit)、数据位(Data Bit)、奇偶校验位(Parity)和停止位(Stop Bit)组成。起始位表示一个字符的开始，接收方可用起始位使自己的接收时钟与数据同步，停止位则表示一个字符的结束。这种用起始位开始、停止位结束所构成的一串信息称为帧(Frame)①。异步通信的传送格式如图 7-3 所示。在传送一个字符时，由一位低电平的起始位开始，接着传送数据位，数据位的位数为 5～8 位。在传送时，按低位在前、高位在后的顺序传送。奇偶校验位用于检验数据传送的正确性(可略)，可由程序指定。最后传送的是高电平的停止位，停止位可以是 1 位、1.5 位或 2 位。停止位结束到下一个字符的起始位之间的空闲位要由高电平“1”来填充(只要不发送下一个字符，线路上就始终为空闲位)。异步通信中典型的帧格式是：1 位起始位，7 位(或 8 位)数据位，

① 异步通信中的“帧”与同步通信中的“帧”是不同的，异步通信中的“帧”只包含一个字符，而同步通信中的“帧”可包含几十个到上千个字符。

1 位奇偶校验位，2 位停止位。

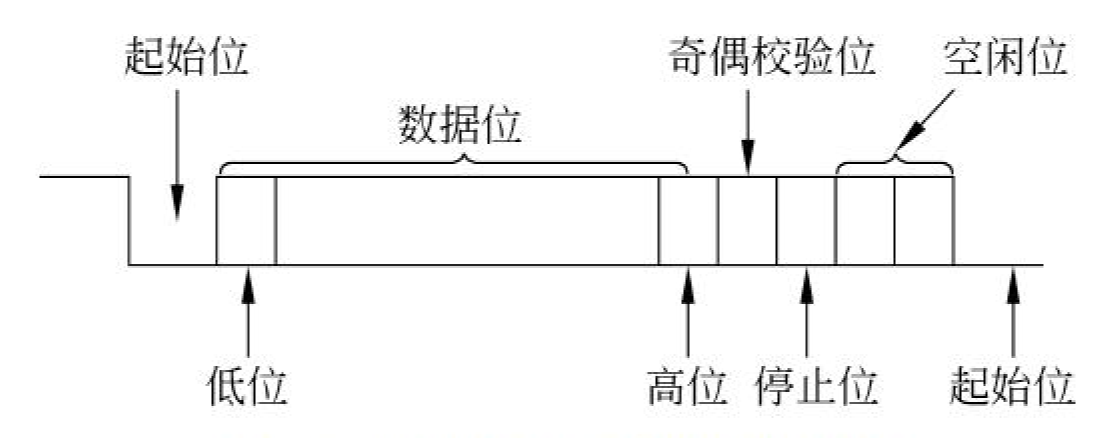


图 7-3 异步串行通信的数据格式

从以上叙述可以看出，在异步通信中，每接收一个字符，接收方都要重新与发送方同步一次，所以接收端的同步时钟信号并不需要严格与发送方同步，只要它们在一个字符的传输时间范围内能保持同步即可。这意味着异步通信对时钟信号漂移的要求要比同步通信低得多，硬件成本也要低得多。但是异步通信每传送一个字符，要增加大约 20％的附加信息位，所以传送效率比较低。异步通信方式简单可靠，也容易实现，故广泛地应用于各种微型机系统中。

4. 串行通信的数据校验

数字通信中一项很重要的技术是差错控制技术，包括对传送的数据自动地进行校验，并在检测出错误时自动校正。对远距离的串行通信，由于信号畸变、线路干扰以及设备质量等问题，有可能会出现传输错误，此时就要求能够自动检测和纠正。目前常用的校验方法有奇偶校验码、循环冗余码等。下面仅简单介绍奇偶校验码。

奇偶校验是一种最简单的校验方法，用于对一个字符的传送过程进行校验。先规定好校验的性质是奇校验还是偶校验。发送时，在每个字符编码的后边增加一个奇偶校验位，其目的是使整个编码(字符编码加上奇偶校验位)中“1”的个数为奇数或者偶数。若编码中“1”的个数为奇数，则为奇校验；否则为偶校验。接收设备在接收时检查所接收到的整个字符编码，看“1”的个数是否符合事先的规定，如果出错，则置错误标志。

奇偶校验只能检查出所传输字符的一位错误，对两位以上同时出错就检查不出来。在实际的传送过程中，一位错的概率在差错中的比例是最大的，同时奇偶校验又比较容易实现，因此，奇偶校验在实际应用中仍非常广泛。目前常用的可编程串行通信接口芯片中都包含有硬件的奇偶校验电路，也可以通过软件编程实现。

循环冗余校验(CRC)是以数据块为对象进行校验的。采用 CRC 码校验要比用奇偶校验码的误码率低几个数量级，它可以把 99.997％以上的各种错误都检查出来。

5. 串行通信的接口标准

串行通信的接口标准有很多，计算机中应用最广泛的是 EIA RS-232-C(Electronics Industry Association Recommended Standard 232-C)接口标准。RS-232-C 规定了接口的机械、电气、功能等方面的参数。

RS-232-C 接口具有以下几个特点。

(1) 信号线少。RS-232-C 接口采用 25 条线，包括两个信号通道，即第一通道(也称

主通道)和第二通道(也称副通道)。利用该接口可实现双工通信。一般主通道较常使用,而副通道使用较少。在通常情况下,双工通信只用很少几条线就可实现。在最简单的情况下,用一条接收线、一条发送线再加一条地线就可实现计算机到计算机或到其他设备的通信。

(2) 有多种可供选择的传送速率,使之能适用于不同速率的设备。RS-232-C 规定的标准传送速率有 50b/s、75b/s、110b/s、150b/s、300b/s、600b/s、1200b/s、2400b/s、4800b/s、9600b/s、19.2Kb/s、33.6Kb/s 和 56Kb/s。

(3) 传送距离远。由于 RS-232-C 采用串行传送方式,并可将 TTL 电平转换为 RS-232-C 的电平,使其传送距离在基带传送时可达 30m。若利用光电隔离 20mA 的电流环进行传送,则其传送距离可达 1000m。当然,若在串行接口上再外接调制解调器,则传送距离就更远。

(4) 采用负逻辑无间隔不归零电平码传送。规定逻辑“1”为−5～−15V 的信号,逻辑“0”为+5～+15V 的信号。逻辑“1”与逻辑“0”之间的电平阈值很大,从而大大提高了抗干扰能力。

7.2 可编程定时/计数器 8253

在数字电路、计算机系统以及实时控制系统中常需要用到定时信号,如函数发生器、计算机中的系统日历时钟、DRAM 的定时刷新、实时采样和控制系统等都要用到定时信号。

定时信号可以利用软件编程或硬件的方法得到。

所谓软件定时的方法,就是设计一个延时子程序,子程序中全部指令执行时间的总和就是该子程序的延时时间。在 CPU 时钟频率一定时,子程序的延时时间是固定的。这种方法比较简单、较易实现,只是需要了解延时子程序中每条指令的执行时间。软件定时的定时时间不太精确,但使用方便,因此在软件开发中经常用到。但它仅适用于延时时间较短、重复次数有限的场合,否则 CPU 总是执行延时程序,占用了大量的时间,使 CPU 的利用率降低。故在对时间要求严格的实时控制系统和多任务系统中很少采用。

硬件定时就是利用专用的硬件定时/计数器,在简单软件控制下产生准确的延时时间。其基本原理是通过软件确定定时/计数器的工作方式、设置计数初值并启动计数器工作,当计数到给定值时便自动产生定时信号。这种方法的成本不高,程序上也很简单,且大大提高了 CPU 的效率,既适合长时间、多次重复的定时,也可用于延时时间较短的场合,因此得到了广泛的应用。

定时/计数器在计数方式上分为加法计数器和减法计数器。加法计数器是每有一个计数脉冲就加 1,当加到预先设定的计数值时产生一个定时信号;减法计数器是在送入计数初值后,每来一个计数脉冲就减 1,减到零时产生一个定时信号输出。可编程定时器 8253 是一个减法计数器,它是 Intel 公司专为 80x86 系列 CPU 配置的外围接口芯片。这里仍然从外部引线入手,介绍 8253 的外部特性和与应用有关的内部结构,最终使读者掌

握芯片与系统的连接和使用方法。

7.2.1 8253的引线及结构

1. 引线及功能

8253是Intel公司生产的三通道16b的可编程定时/计数器，是具有24根引脚的双列直插式器件，其外部引线如图7-4所示。它的最高计数频率可达2MHz，使用单电源+5V供电，输入输出均与TTL电平兼容，其主要引脚的功能如下。

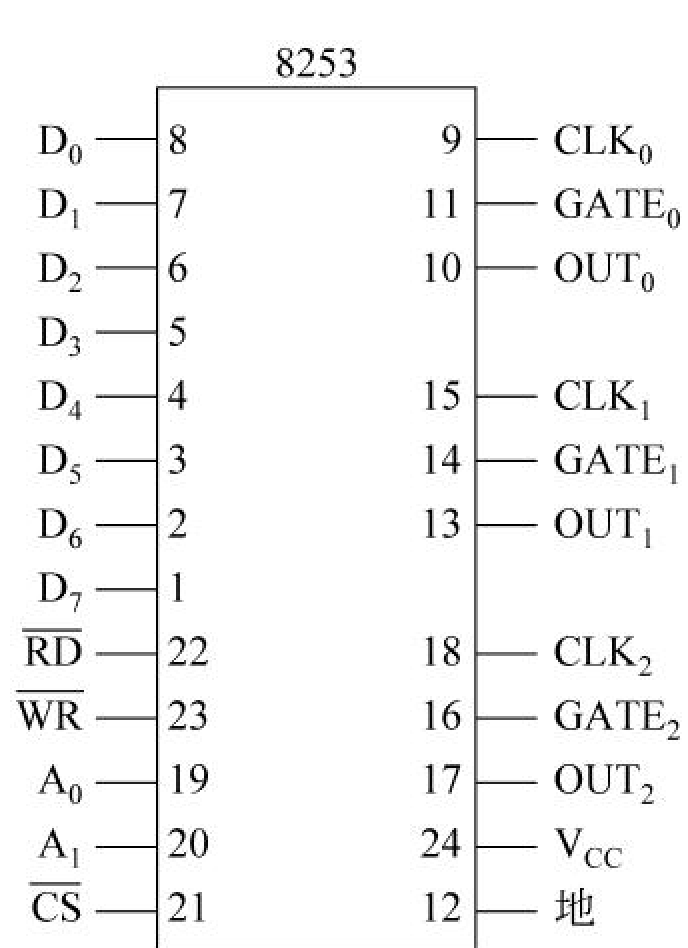


图7-4 可编程定时器8253外部引线图

(1) D_0～D_7：8位双向数据线。D_0～D_7用来传送数据、控制字和计数器的计数初值。

(2) $\overline{CS}$：片选信号，输入，低电平有效。由系统高位I/O地址译码产生。当它有效时，此定时器芯片被选中。

(3) $\overline{RD}$：读控制信号，输入，低电平有效。当它有效时表示CPU要对此定时器芯片进行读操作。

(4) $\overline{WR}$：写控制信号，输入，低电平有效。当它有效时表示CPU要对此定时器芯片进行写操作。

(5) A_0、A_1：地址信号线。高位地址信号经译码产生$\overline{CS}$片选信号，决定了8253芯片所具有的地址范围。而A_0和A_1地址信号则经片内译码产生4个有效地址，分别对应了芯片内部3个独立的计数器(通道)和一个控制寄存器。具体规定如表7-1所示。

表7-1 各地址信号组合功能

A_1	A_0	功 能	A_1	A_0	功 能
0	0	选择计数器0	1	0	选择计数器2
0	1	选择计数器1	1	1	选择控制寄存器

(6) CLK_0～CLK_2：每个计数器的时钟信号输入端。计数器对此时钟信号进行计数。CLK信号是计数器工作的计时基准，因此其频率要求很精确。

(7) $GATE_0$～$GATE_2$：门控信号，用于控制计数的启动和停止。多数情况下，GATE=1时允许计数，GATE=0时停止计数。但有时仅用GATE的上升沿启动计数，启动后则GATE的状态不再影响计数过程。这在7.2.2节将会详细介绍。

(8) OUT_0～OUT_2：计数器输出信号。在不同的工作方式下OUT_0～OUT_2将产生不同的输出波形。

2. 内部结构和工作原理

图7-5所示为8253的内部结构示意图，它主要包括3个计数器、1个控制寄存器以及数据总线缓冲器和读写逻辑电路。

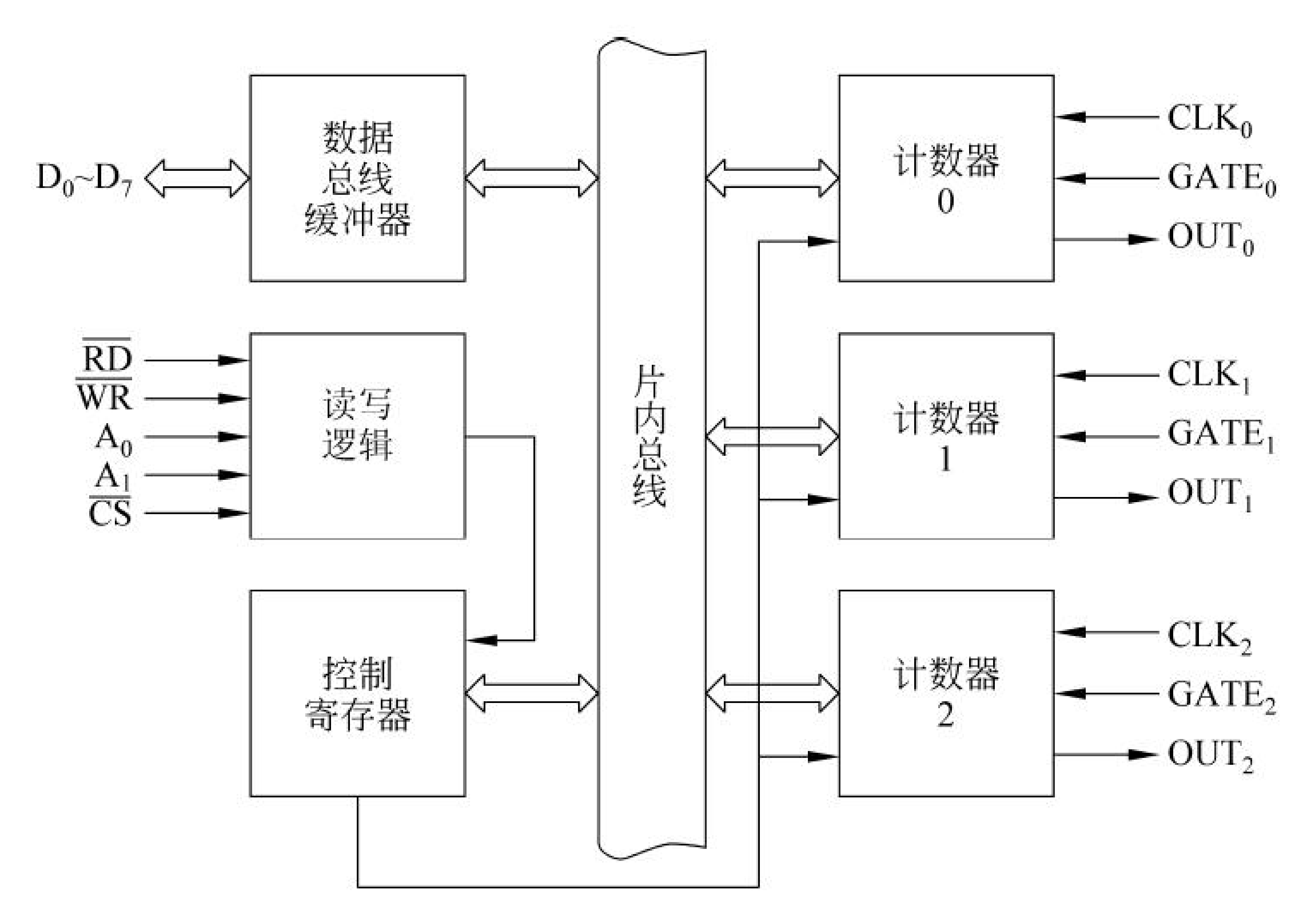


图 7-5　可编程定时器 8253 的内部结构框图

1）计数器

计数器 0（CNT_0）、计数器 1（CNT_1）和计数器 2（CNT_2）是 3 个相同的 16 位计数器，它们相互独立，可以分别按各自的方式进行工作，每个计数器都包括一个 16 位的初值寄存器、一个计数执行单元和一个输出锁存器。其工作过程如下。

当置入初值后，计数执行单元开始对输入脉冲 CLK 进行减 1 计数，在减到零时，从 OUT 端输出一个信号，整个过程可以重复进行。计数器既可按二进制计数，也可按十进制计数。另外，在计数过程中，计数器还受到门控信号 GATE 的控制。在不同的工作方式下，计数器的输入 CLK、输出 OUT 和门控信号 GATE 之间的关系将会不同（详情见 7.2.2 节）。

2）控制寄存器

8253 是可编程接口芯片，可以通过软件编程写入控制字的方法控制其工作方式。芯片内部的控制寄存器就是用来存放控制字的。控制字在 8253 初始化时通过输出指令写入控制寄存器。该寄存器为 8 位，只能写入，不能读出。

3）数据总线缓冲器

数据总线缓冲器是一个 8 位的双向三态缓冲器，用于 8253 和 CPU 数据总线之间连接的接口。CPU 通过该数据缓冲器对 8253 进行读写。

4）读写控制逻辑

在片选信号 $\overline{CS}$ 有效的情况下，读写控制逻辑从系统总线接收输入信号，经过逻辑组合，产生对各部分的控制信号。当片选信号 $\overline{CS}$ 无效，即 $\overline{CS}$ 为高电平时，数据总线缓冲器处于三态，读写信号得不到确认，CPU 则无法对其进行读写操作。

3. 计数启动方法

8253 计数器的计数过程可以由程序指令启动，称为软件启动；也可由外部电路信号启动，称为硬件启动。

1）软件启动

软件启动在 CPU 用输出指令向计数器写入初值后就启动计数。但事实上，CPU 写入的计数初值只是写到了计数器内部的初值寄存器中，计数过程并未真正开始。写入初值后的第一个 CLK 信号将初值寄存器中的内容送到计数器中，而从第二个 CLK 脉冲的下降沿开始，计数器才真正进行减 1 计数。之后，每来一个 CLK 脉冲都会使计数器减 1，直到减到零时在 OUT 端输出一个信号。因此，从 CPU 执行输出指令写入计数初值到计数结束，实际的 CLK 脉冲个数比编程写入的计数初值 N 要多一个，即($N+1$)个。只要是用软件启动计数，这种误差是不可避免的。

2）硬件启动

硬件启动在写入计数初值后并不启动计数，而是在门控信号 GATE 由低电平变高后，再经 CLK 信号的上升沿采样，之后在该 CLK 的下降沿才开始计数。由于 GATE 信号与 CLK 信号不一定同步，故在极端情况下，从 GATE 变高到 CLK 采样之间的延时可能会经历一个 CLK 脉冲宽度，因此在计数初值与实际的 CLK 脉冲个数之间也会有一个误差。

在多数工作方式下，计数器每启动一次只工作一个周期(即从初值减到零)，要想重复计数过程则必须重新启动，因此称它们为不自动重复的计数方式。除此之外，8253 还有另外一种计数方式，即一旦计数启动，只要门控信号 GATE 保持高电平，计数过程就会自动周而复始地重复下去，这时 OUT 端可以产生连续的波形输出，称这种计数过程为自动重复的计数方式，此时，在达到稳定状态后，上面讲到的因启动造成的实际计数值和计数初值之间的误差就不再存在。

7.2.2 8253 的工作方式

8253 共有 6 种不同的工作方式，在不同的工作方式下，计数过程的启动方式、OUT 端的输出波形都不一样，自动重复功能和 GATE 的控制作用以及写入新的计数初值对计数过程产生的影响也不相同。下面将借助工作波形来分别说明这 6 种工作方式的计数过程。

1. 方式 0——计数结束中断

方式 0 为软件启动、不自动重复计数的方式。在这种方式下，在第一个写信号$\overline{WR}$有效时向计数器写入控制字 CW，之后其输出端 OUT 就变低电平；在第二个$\overline{WR}$有效时装入计数初值，然后经过一个 CLK 信号的上升沿和下降沿，初值进入计数器；当计数减到零——计数结束后，OUT 输出变为高电平，波形如图 7-6 所示。该输出信号可作为中断请求信号使用。

不自动重复计数的特点是：每写入一次计数初值只计数一个周期，若要重新计数，需 CPU 再次写入计数初值。

有以下两点需要注意。

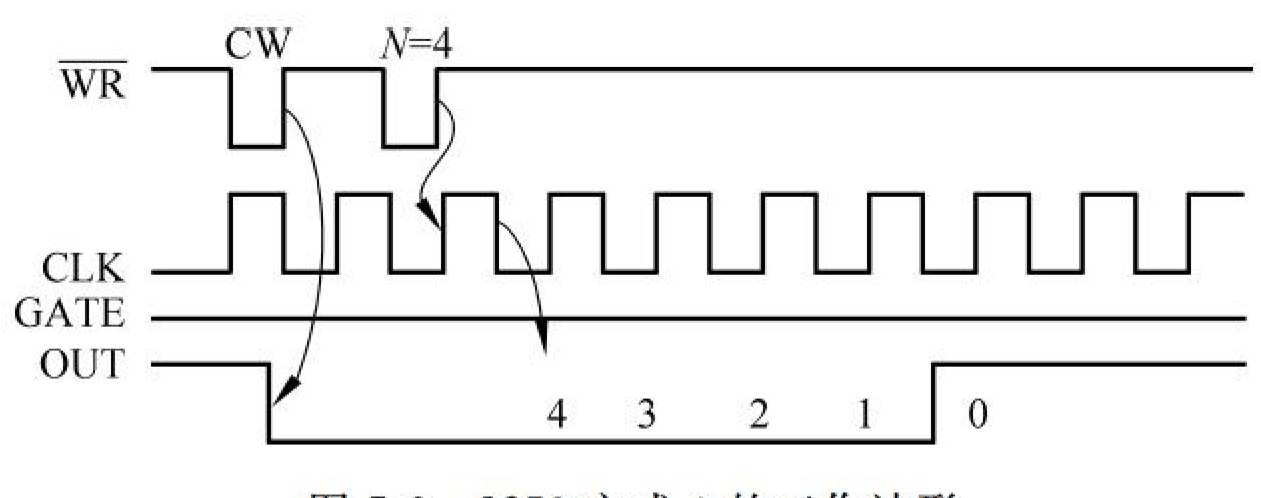


图 7-6　8253 方式 0 的工作波形

(1) 整个计数过程中，GATE 端应始终保持为高电平。若 GATE 变低，则暂停计数，直到 GATE 变高后再接着计数。

(2) 计数过程中可随时修改计数初值，即使原来的计数过程没有结束，计数器也用新的计数初值重新计数。但如果新的计数初值是 16 位的，则在写入第一个字节后停止原先的计数，写入第二个字节后才开始以新的计数值重新计数。

2. 方式 1——可重复触发的单稳态触发器

方式 1 是一种硬件启动、不自动重复的工作方式。当写入方式 1 的控制字后，OUT 端输出高电平。在 CPU 写入计数初值后，计数器并不开始计数，而是要等门控信号 GATE 出现由低到高的跳变(触发)后，在下一个 CLK 脉冲的下降沿才开始计数，此时 OUT 端立刻变为低电平。当计数结束后，OUT 端输出高电平。这样就可以从计数器的 OUT 端得到一个负脉冲，负脉冲宽度为计数初值 N 乘以 CLK 的周期 T_{CLK}。

方式 1 的特点如下。

(1) 计数过程一旦启动，GATE 端即使变低也不会影响计数。

(2) 可重复触发。当计数到 0 后，不用再次写入计数初值，只要用 GATE 的上升沿重新触发一次计数器，即可产生一个同样宽度的负脉冲。

(3) 在计数过程中，若写入新的计数值，则本次计数过程的输出不受影响。本次计数结束后再次触发，计数器才开始按新的计数值进行计数，并按新值输出脉冲宽度。

(4) 若在形成单个负脉冲的计数过程中外部的 GATE 上升沿提前到来，则下一个 CLK 脉冲的上升沿使计数器重新装入计数初值，并紧接着在 CLK 的下降沿重新开始计数。这时的负脉冲宽度将会加宽，宽度为重新触发前的已有的宽度与新一轮计数过程的宽度之和。方式 1 的波形如图 7-7 所示。

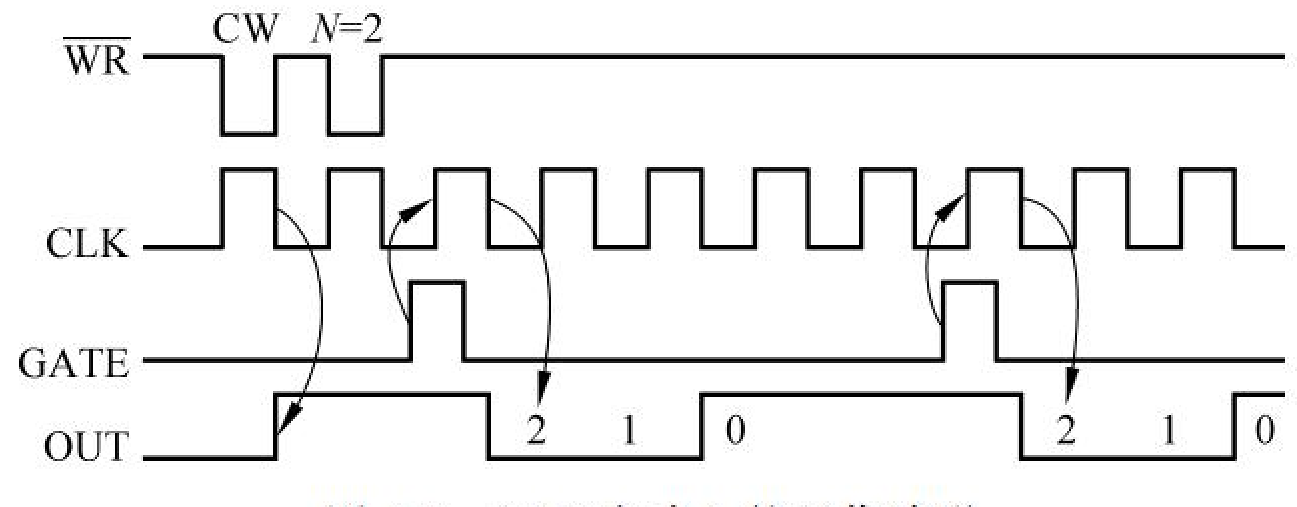


图 7-7　8253 方式 1 的工作波形

3. 方式 2——频率发生器

在方式 2 下，计数器既可以用软件启动，也可以用硬件启动。若写入控制字和计数初值期间 GATE 一直为高电平，则在写入计数初值后的下一个 CLK 开始计数(即软件启动)；若送计数初值时 GATE 为低电平，则要等到 GATE 信号由低变高时才启动(即硬件启动)。一旦计数启动，计数器可以自动重复工作。

在写入方式 2 控制字后，OUT 端变为高电平。假设此时 GATE＝1，则装入计数初值 N 后计数器从下一个 CLK 的下降沿开始计数，经过($N-1$)个 CLK 周期后(此时计数值减为 1)，OUT 端变为低电平，再经过一个 CLK 周期，计数值减到零，OUT 又恢复为高电平。由于方式 2 下计数器可自动重复计数，因此在计数减到零后，计数器又自动装入计数初值，并开始新的一轮计数过程。这样，在 OUT 端就会连续输出宽度为 T_{CLK} 的负脉冲，其周期为 $N\times T_{CLK}$，即 OUT 端输出的脉冲频率为 CLK 的 $1/N$。所以方式 2 也称为分频器，分频系数就是计数初值 N。可以利用不同的计数初值实现对 CLK 时钟脉冲进行 1～65 536 的分频。方式 2 的工作波形如图 7-8 所示。

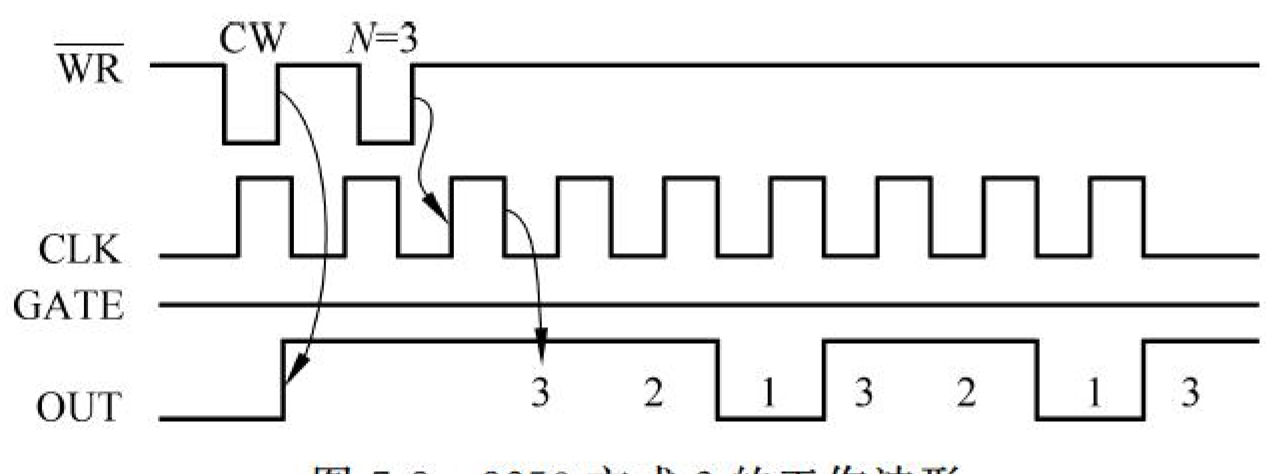


图 7-8　8253 方式 2 的工作波形

在方式 2 中，门控信号 GATE 可被用作控制信号。当 GATE 为低电平时，计数停止，强迫 OUT 输出高电平。当 GATE 变高后的下一个时钟下降沿，计数器又被置入初值从头开始重新计数，之后的过程就和软件启动相同。这个特点可用于实现计数器的硬件同步。

在计数过程中，若重新写入新的计数初值，则不影响当前的计数过程，而是在下一轮计数过程才按新的计数值进行计数。

方式 2 中，一个计数周期应包括 OUT 输出的负脉冲所占的那一个时钟周期。

4. 方式 3——方波发生器

方式 3 和方式 2 类似，也有两种启动方式，也能够自动重复计数。只是计数到 $N/2$ 时，OUT 端输出变为低电平，再接着计数到 0 时，OUT 又变为高，并开始新一轮计数。此时 OUT 端输出的波形不是负脉冲，而是方波。图 7-9 为方式 3 的工作波形。

由图可以看出，在写入方式 3 的控制字 CW 后，OUT 端立刻变高电平。若此时 GATE＝1，则装入计数初值 N 后开始计数。如果装入的计数值 N 为偶数，则计数到 $N/2$ 时，OUT 变低，计完其余的 $N/2$ 后，OUT 又回到高电平。如此这般自动重复下去，OUT 端输出周期为 $N\times T_{CLK}$ 的对称方波。

若 N 为奇数，则输出波形不对称，其中($N+1$)/2 个时钟周期，OUT 为高电平，而另

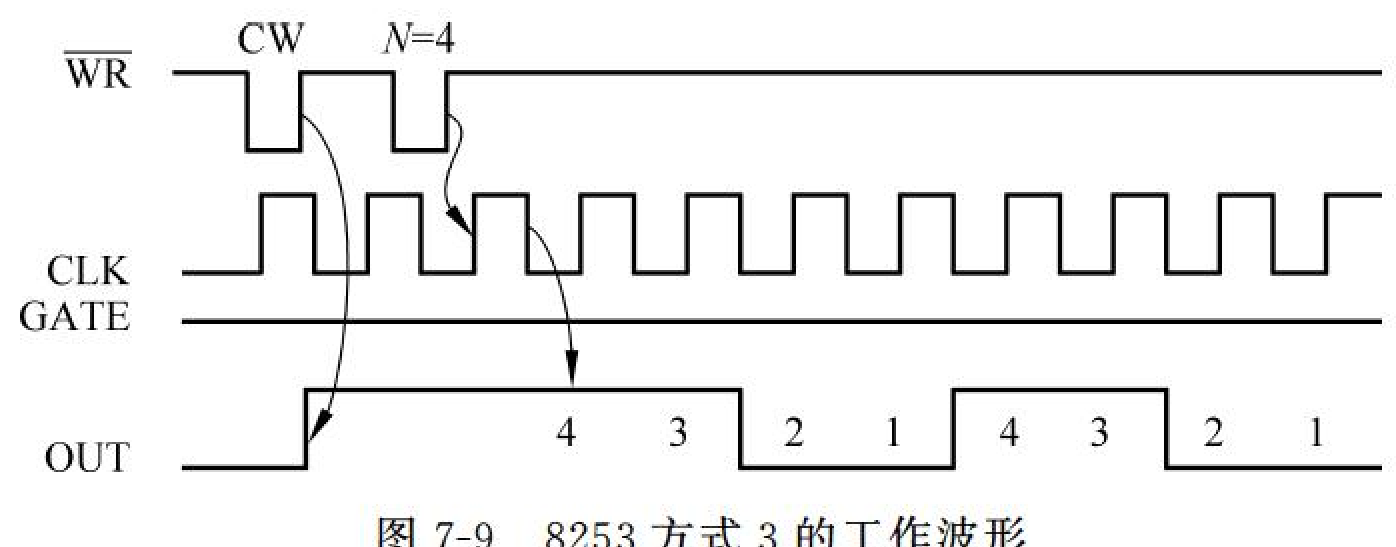


图 7-9　8253 方式 3 的工作波形

外$(N-1)/2$个时钟周期，OUT 为低电平。

写入计数初值时，若 GATE 信号为低电平，则并不开始计数，OUT 端强迫输出高电平。直到 GATE 变为高电平后才启动计数，输出对称方波。若计数过程中 GATE 变低，会立刻终止计数，且 OUT 端马上变高。当 GATE 恢复高电平后，计数器将重新装入计数初值，从头开始计数。在计数过程中，若装入新的计数值，会在当前半周期结束时启用新的计数初值。当然，如果在改变计数初值后接着又发生硬件启动，则会立即以新计数值开始计数。

5. 方式 4——软件触发选通

方式 4 为软件启动、不自动重复计数的方式。写入方式 4 控制字后，输出 OUT 立即变高电平。若 GATE＝1，则装入计数初值后计数立即开始。计数结束时，由 OUT 输出一个 CLK 周期宽的负脉冲。方式 4 的工作波形如图 7-10 所示。

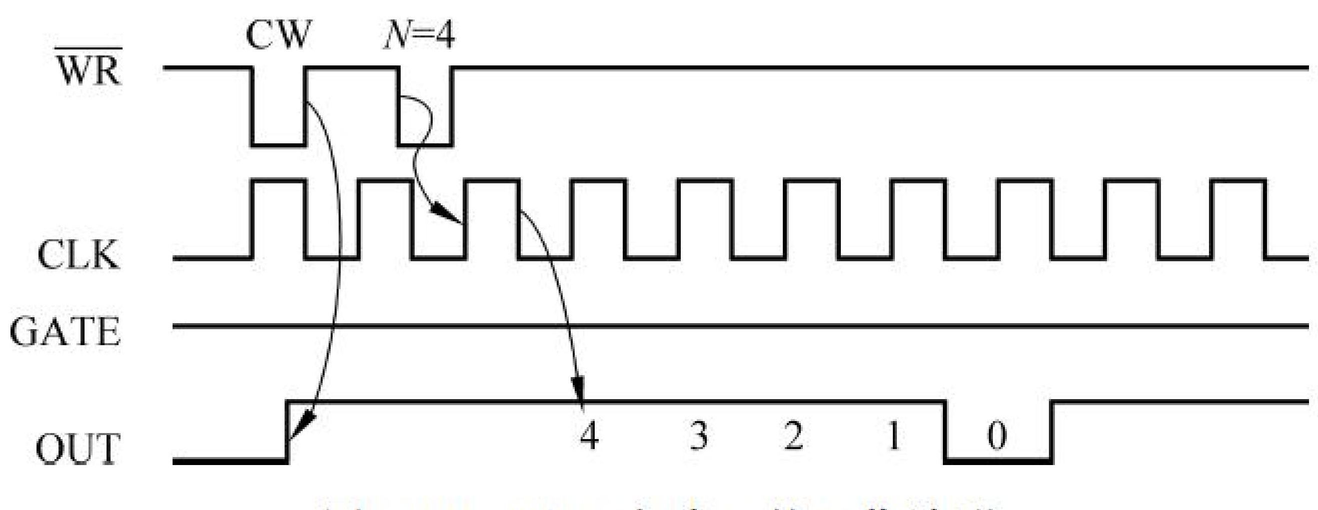


图 7-10　8253 方式 4 的工作波形

该方式下计数器工作的特点与方式 0 相似。如果在计数过程中装入新的计数值，则计数器从下一时钟周期开始按新的计数值重新开始计数。

请注意方式 4 与方式 2 下 OUT 端输出波形的不同。

6. 方式 5——硬件触发选通

方式 5 为硬件启动、不自动重复计数的计数方式（与方式 1 相同）。写入方式 5 控制字后，输出 OUT 变高电平。当 GATE 端出现一个上升沿跳变时，启动计数，计数结束时 OUT 端送出一个宽度为 T_{CLK} 的负脉冲，之后，OUT 又变高且一直保持到下一次计数结束。方式 5 的工作波形如图 7-11 所示。

为便于读者比较，表 7-2 列出了 8253 计数器 6 种工作方式的特点。

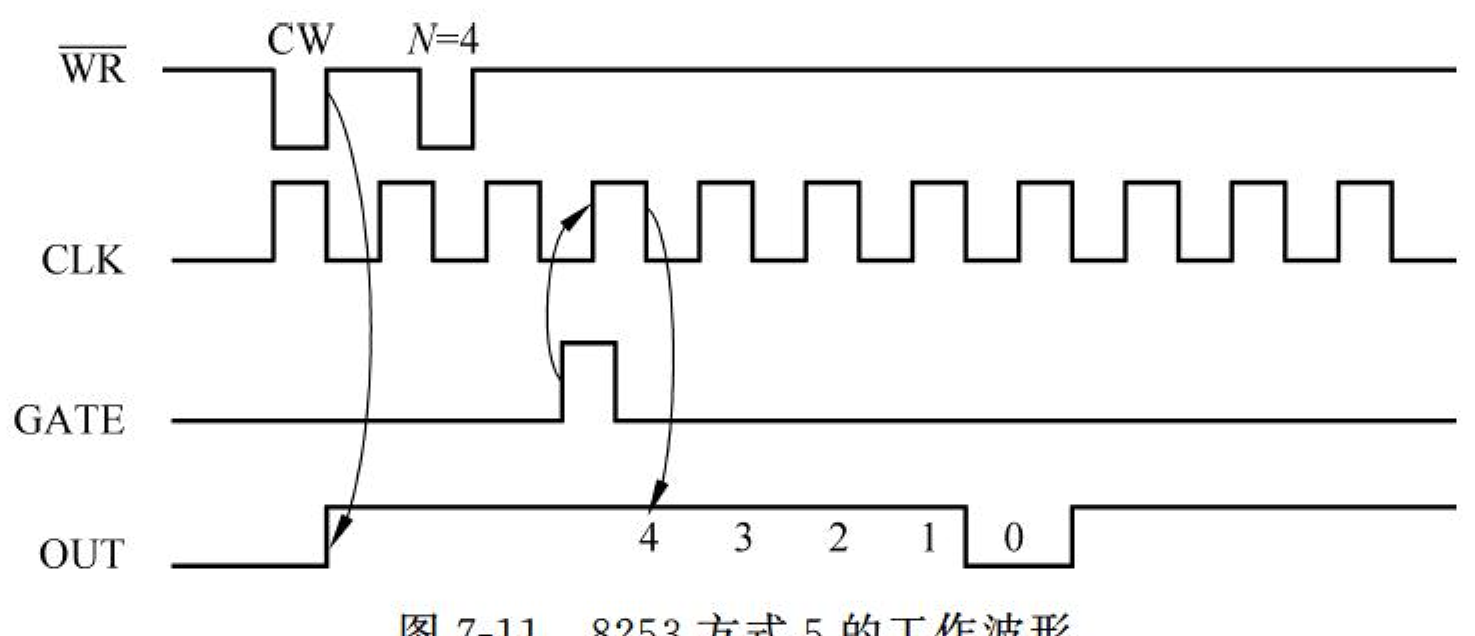


图 7-11　8253 方式 5 的工作波形

表 7-2　8253 计数器工作方式一览表

工作方式	启动计数	中止计数	自动重复	更新初值	输 出 波 形
0	软件	GATE=0	否	立即有效	延时时间可变的上升沿
1	硬件	—	否	下一轮有效	宽度为 $N \times T_{CLK}$ 的单一负脉冲
2	软/硬件	GATE=0	是	下一轮有效	周期为 $N \times T_{CLK}$，宽度为 T_{CLK} 的连续负脉冲
3	软/硬件	GATE=0	是	下半轮有效	周期为 $N \times T_{CLK}$ 的连续方波
4	软件	GATE=0	否	立即有效	宽度为 T_{CLK} 的单一负脉冲
5	硬件	—	否	下一轮有效	宽度为 T_{CLK} 的单一负脉冲

7.2.3　8253 的控制字

8253 必须先初始化才能正常工作，每个计数通道可分别初始化。CPU 通过指令将控制字写入可编程定时器 8253 的控制寄存器，从而确定 3 个计数器分别工作于何种工作方式下。8253 的控制字具有固定的格式，如图 7-12 所示。

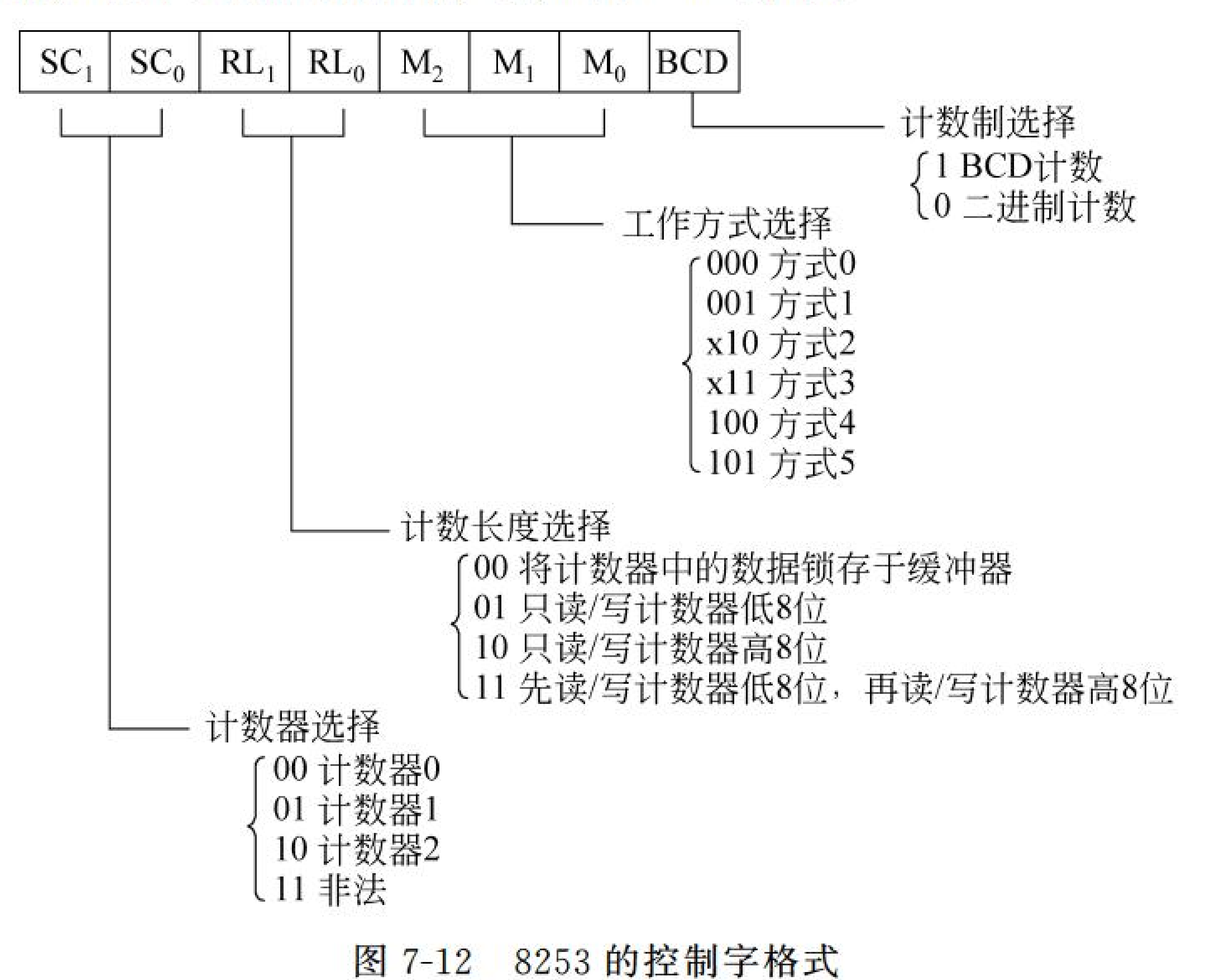


图 7-12　8253 的控制字格式

控制字的 D_0 位用来定义用户所使用的计数值是二进制数还是 BCD 数。因为每个计数器的字长都是 16 位，所以如果采用二进制计数则计数范围为 0000H～FFFFH；而如果用 BCD 计数，则计数范围为 0000～9999。由于计数器做减 1 操作，故当计数初值为 0000 时，对应的是最大计数值（二进制计数时为 65536，十进制计数时为 10000）。

在 8253 计数过程中，CPU 可随时读出其当前的计数值，而且不会影响计数器的工作。实现这种操作只需写入相应的控制字，此时控制字的 RL_1RL_2 选择 00，即控制字格式为 $SC_1SC_0$0 0 ××××。控制字其他各位的功能图中标得都很清楚，这里就不再说明。

7.2.4 8253 的应用

1. 8253 与系统的连接

8253 共占用了 4 个端口地址，地址范围由高位地址信号决定，高位地址的译码输出接到片选端 $\overline{CS}$，A_0 和 A_1 分别接到系统总线的 A_0、A_1 地址信号线上，用来寻址芯片内部的 3 个计数器及控制寄存器。信号 $\overline{CS}$、A_0、A_1 与读信号 $\overline{RD}$、写信号 $\overline{WR}$ 配合，可以实现对 8253 的各种读写操作。上述各信号的功能组合如表 7-3 所示。

表 7-3 各寻址信号组合功能

$\overline{CS}$	A_1	A_0	$\overline{RD}$	$\overline{WR}$	功能	$\overline{CS}$	A_1	A_0	$\overline{RD}$	$\overline{WR}$	功能
0	0	0	1	0	写计数器 0	0	0	0	0	1	读计数器 0
0	0	1	1	0	写计数器 1	0	0	1	0	1	读计数器 1
0	1	0	1	0	写计数器 2	0	1	0	0	1	读计数器 2
0	1	1	1	0	写控制寄存器	0	1	1	0	1	无效

对 8253 的读写操作需注意以下两点。

(1) 在向某一计数器写入计数初值时，应与控制字中 RL_1 和 RL_0 的编码相对应。当编码为 01 或 10 时，只可写入一个字节的计数初值，另一字节 8253 默认为 0；当编码为 11 时，一定要装入两个字节的计数值，且先写入低字节再写入高字节。若此时只写了一个字节就去写别的计数器的计数值，则写入的字节将被解释为计数值的高 8 位，从而产生错误。

(2) 8253 的计数器在计数过程中可读出其当前计数值。读出的方法有两种：①前面已讲到的在计数过程中读计数值的方法，即写入 RL_1 和 RL_0 为 00 的控制字，将选中的计数器的当前计数值锁存到相应锁存器中，而后利用读计数器操作——用两条输入指令即可把 16 位计数值读出；②控制 GATE 门控信号使计数器停止计数，先写入控制字，规定好 RL_1 和 RL_0 的状态，也就是规定读一个字节还是读两个字节。若其编码为 11，则一定要读两次，先读出计数值低 8 位，再读出高 8 位。此时若读一次同样会出错。

可编程定时器 8253 可直接连接到系统总线上。图 7-13 就是 8253 与 8088 系统总线连接的一个例子。该图中，系统地址总线信号 A_{15}～A_2 经译码电路译码产生片选信号选中 8253，8253 占用的 4 个端口地址为 FF04H～FF07H。

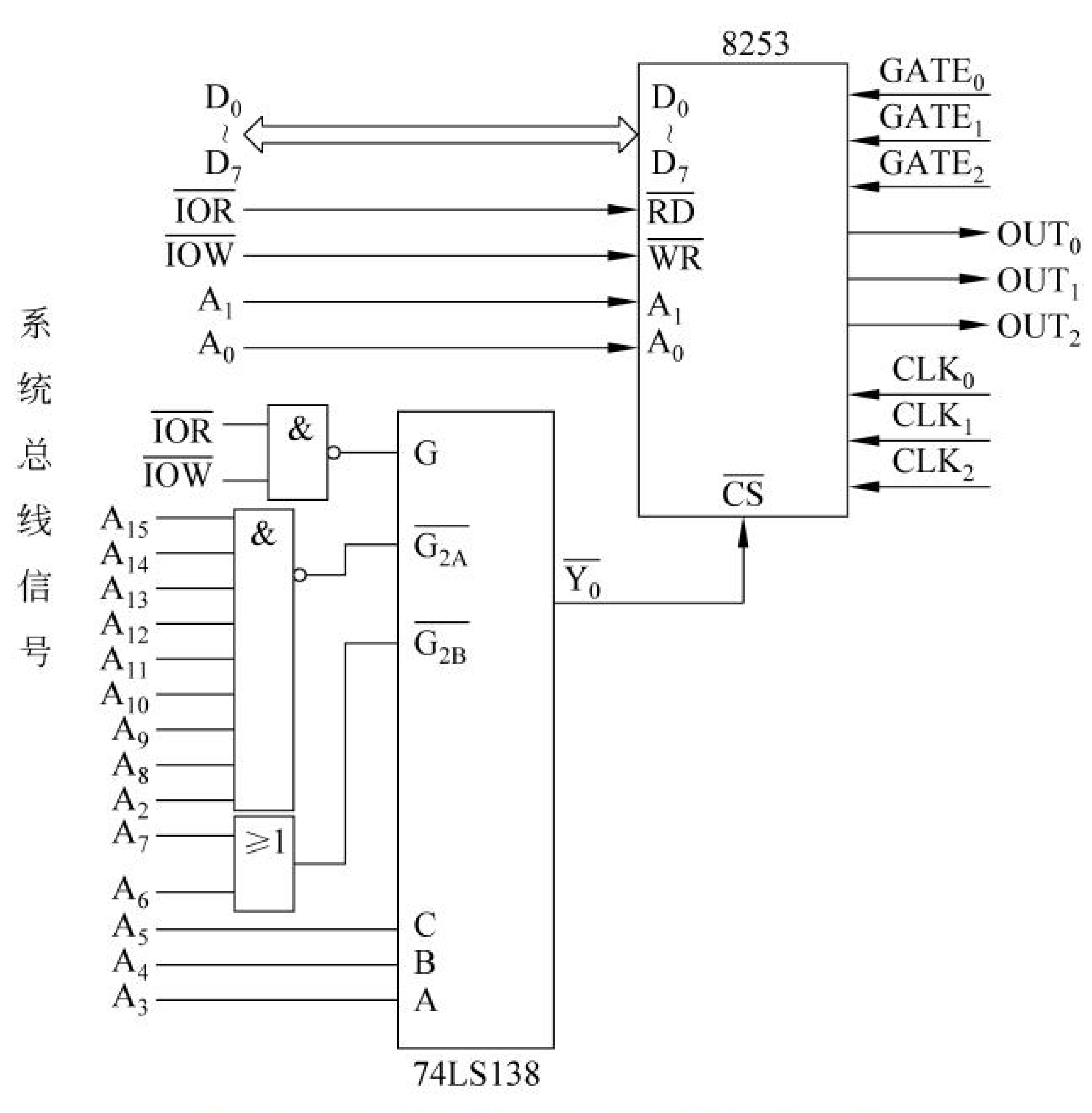


图 7-13　8253 与 8088 系统总线的连接

2. 8253 的编程

对 8253 的编程也称为对 8253 进行初始化。它包括两部分：写各计数器的方式控制字、设置计数初值。由于 8253 每个计数器都有自己的地址，控制字中又有专门两位来指定计数器，这使得对计数器的初始化可按任何顺序进行。初始化的方法可以有以下两种。

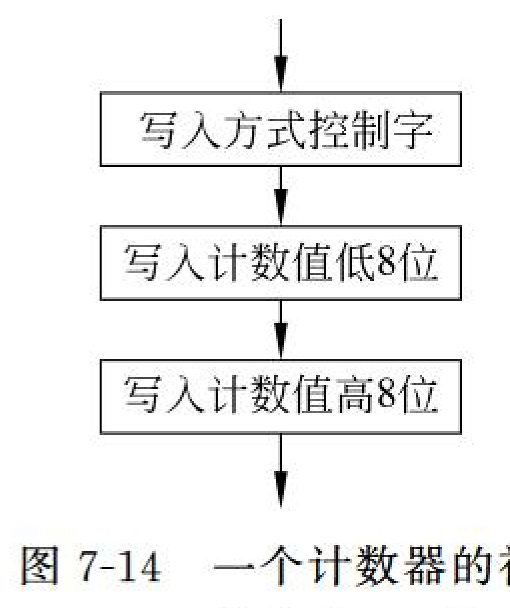


图 7-14　一个计数器的初始化编程顺序

(1) 以计数器为单位逐个进行初始化，即对某一个计数器，先写入方式控制字，接着写入计数初值(一个字节或两个字节)。先初始化哪一个计数器无关紧要，但对某一个计数器来说，则必须按照“方式控制字—计数值低字节—计数值高字节”的顺序进行初始化，如图 7-14 所示。

(2) 先写所有计数器的方式字，再装入各计数器的计数值，这种方法的过程如图 7-15 所示。从图可以看出，这种初始化方法是先分别写入各计数器的方式控制字，再分别写入计数初值，计数初值仍要按先低字节再高字节的顺序写入。

由于输入输出指令的要求，在写入计数初值时，设定的计数值必须在累加器 AL 中。但双字节计数时，计数初值设定在 AX 中，所以要求在写高 8 位时，要将 AH 内容送 AL，然后再写入控制寄存器。这一点在下面的例子中要注意。

对以上两种初始化方法，读者可根据自己的习惯采用任意一种。

【例 7-1】 在 IBM PC 系统板上使用了一片 8253 定时/计数器，其计数器 0(CNT_0)用于为系统的电子钟提供时间基准，它的输出端作为系统的中断源接到 8259 的 IR_0 端；计数器 1(CNT_1)用于 DRAM 的定时刷新；计数器 2(CNT_2)主要用作机内扬声器的音频

信号源，可输出不同频率的方波信号。图 7-16 是简化了的 IBM PC 内 8253 的连接图，其接口地址采用部分译码方式，占用的设备端口地址为 40H～5FH。以下编程中，使用了地址中的 40H～43H。

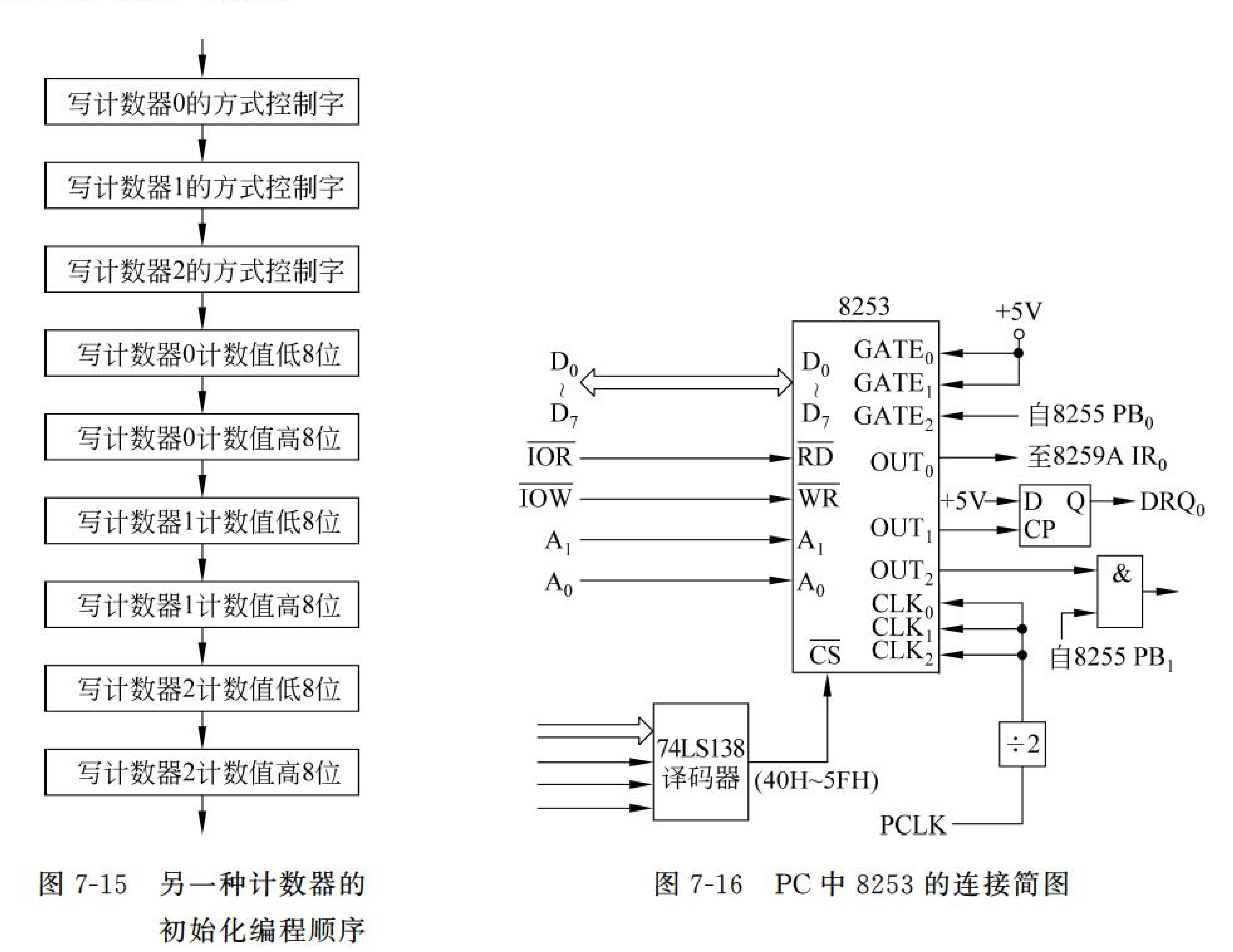


图 7-15　另一种计数器的初始化编程顺序

图 7-16　PC 中 8253 的连接简图

3 个计数器的输入时钟频率均为 1.19MHz。

由于计数器 0 的作用是为系统提供时间基准，将其初始化为方式 3，产生周期的方波信号，计数初值选为最大计数值，即十六进制的 0000H(65536)。根据方式 3 的工作原理可知，OUT_0 输出方波信号的频率为 1.19MHz/65536≈18.2Hz。由于 OUT_0 与 8259A 的中断请求输入线 IR_0 相连接，所以每秒将会产生 18.2 次中断请求，该中断请求用于维护系统的日历钟。

CNT_1 初始化为方式 2，计数初值取 18，18/1.19MHz≈15μs，即每 15μs 对动态存储器刷新一次。

CNT_2 初始化为方式 3，控制扬声器发出频率为 1kHz 的声音，故取时间常数为 1190。在 PC 中，要使扬声器发声，还必须使 8255 的 PB_1 和 PB_0 输出高电平(设 8255 的 B 口地址为 61H)。

IBM PC 中 8253 的初始化程序如下：

```
;CNT0 初始化
    MOV  AL, 36H              ;选择计数器 0,写双字节计数值,方式 3,二进制计数
    OUT  43H, AL              ;控制字写入控制寄存器
    MOV  AL, 0                ;选最大计数值 (65536)
    OUT  40H, AL              ;写低 8位计数值
```

```
    OUT   40H, AL              ;写高 8 位计数值
;CNT1 初始化
    MOV   AL, 54H              ;选择计数器 1,低 8 位单字节计数值,方式 2,二进制计数
    OUT   43H, AL
    MOV   AL, 18
    OUT   41H, AL              ;计数值写入计数器 1
;CNT2 初始化
    MOV   AL, 0B6H             ;选择计数器 2,双字节计数值,方式 3,二进制计数
    OUT   43H, AL
    MOV   AX, 1190
    OUT   42H, AL              ;送低字节到计数器 2
    MOV   AL, AH               ;(AL)←高字节计数值
    OUT   42H, AL              ;高 8 位计数值写入计数器 2
    IN    AL, 61H              ;读 8255 的 B 口
    MOV   AH, AL               ;将 B 口内容保存
    OR    AL, 03               ;使 PB0=PB1=1
    OUT   61H, AL              ;使扬声器发声
    ⋮
    MOV   AL, AH               ;恢复 8255B 口状态
    OUT   61H, AL
```

【例 7-2】 写出图 7-13 中 8253 的初始化程序。其中,3 个 CLK 频率均为 2MHz,计数器 0 在定时 100μs 后产生中断请求;计数器 1 用于产生周期为 10μs 的对称方波;计数器 2 每 1ms 产生一个负脉冲。

根据要求可知,计数器 0 应工作于方式 0,计数初值=100μs/0.5μs=200(CLK 的周期=0.5μs);计数器 1 应工作于方式 3,计数初值=10μs/0.5μs=20;计数器 2 应工作于方式 2,计数初值=1ms/0.5μs=2000。以下是 8253 的初始化程序。

```
START: MOV   DX, 0FF07H
       MOV   AL, 10H           ;计数器 0,只写计数值低 8 位,方式 0,二进制计数
       OUT   DX, AL
       MOV   AL, 56H           ;计数器 1,只写计数值低 8 位,方式 3,二进制计数
       OUT   DX, AL
       MOV   AL, 0B4H          ;计数器 2,先写低 8 位再写高 8 位,方式 2,二进制计数
       OUT   DX, AL
       MOV   DX, 0FF04H
       MOV   AL, 200           ;计数器 0 的计数初值
       OUT   DX, AL
       MOV   DX, 0FF05H
       MOV   AL, 20            ;计数器 1 的计数初值
       OUT   DX, AL
       MOV   DX, 0FF06H
       MOV   AX, 2000          ;计数器 2 的计数初值
       OUT   DX, AL
       MOV   AL, AH
```

```
OUT  DX, AL
…
```

从以上的叙述中可以看到，8253 在应用上具有很高的灵活性，通过对外部输入时钟信号的计数可以达到计数和定时两种应用目的。门控信号 GATE 提供了从外部控制计数器的能力。同时，当一个计数器计数或定时长度不够时，还可以把两个、三个计数器串联起来使用，即一个计数器的输出 OUT 作为下一个计数器的外部时钟 CLK 输入，甚至可将两个 8253 串起来使用。这些方面的问题，只要读者熟悉了 8253 的基本功能就不难举一反三更巧妙地使用它。

回到我们的"家庭安全防盗系统"。由于 8253 定时/计数器在工作于方式 3 时，可以输出连续方波信号，因此可以利用其作为报警控制信号。

"家庭安全防盗系统"设计方案示例 2：

设计同样基于方案示例 1 给出的假设，即监测装置输出电平信号。当出现异常时，监测装置输出高电平（"1"），正常状态则输出低电平（"0"）。

基本方案：利用三态门接口读取 8 个监测装置的输出信息，利用 8253 通道 0 控制报警器发声，利用锁存器接口控制报警灯闪烁以及 8253 通道 0 的启动。可以设计如图 7-17 所示的系统连接示意图。

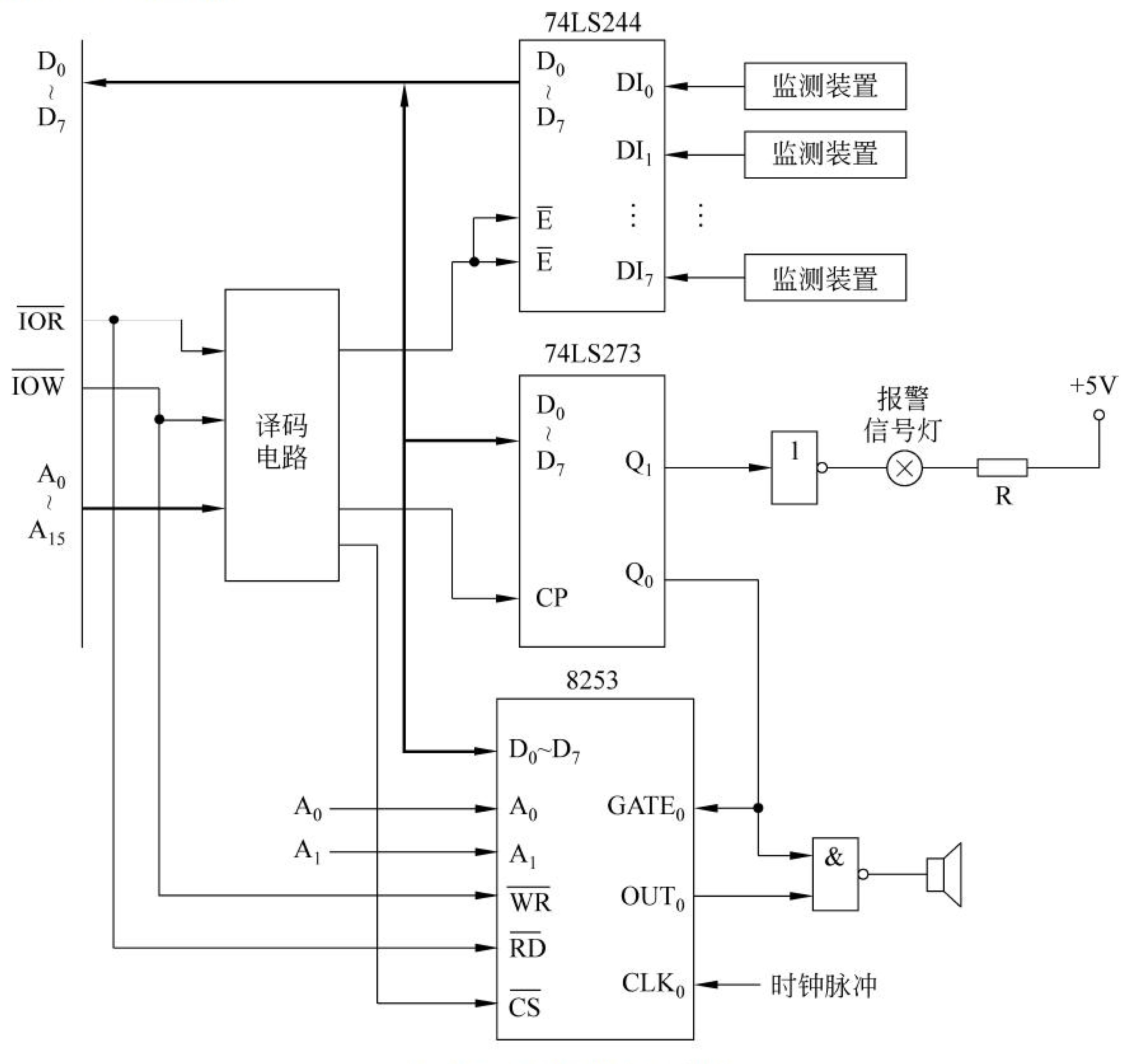


图 7-17　系统连接示意图

请试一下，完善整个设计。

7.3 可编程并行接口8255

并行接口是实现并行通信的接口。其数据传送方向有两种：单向传送（只作为输入口或输出口）、双向传送（既可作为输入口，也可作为输出口）。并行接口可以很简单，如锁存器或三态门；也可以很复杂，如可编程并行接口芯片。7.3节所介绍的8255是Intel公司生产的为x86系列CPU配套的可编程并行接口芯片。所谓可编程，就是可以通过软件的方式来设定芯片的工作方式。8255的通用性较强、使用灵活，是一种典型的可编程并行接口。

7.3.1 8255的引线及结构

1. 外部引线及结构

8255的外部引线如图7-18所示，共有40个引脚，其功能如下。

（1）$D_0 \sim D_7$：双向数据信号线，用来传送数据和控制字。

（2）$\overline{RD}$：读信号线，低电平有效。$\overline{RD}$与其他信号线一起实现对8255接口的读操作，通常接系统总线的$\overline{IOR}$信号。

（3）$\overline{WR}$：写信号线，低电平有效。$\overline{WR}$与其他信号一起实现对8255的写操作，通常接系统总线的$\overline{IOW}$。

（4）$\overline{CS}$：片选信号线，低电平有效。当系统地址信号经译码产生低电平时选中8255芯片，使能够对8255进行操作。

（5）A_0、A_1：口地址选择信号线。

8255的内部包括3个独立的输入输出端口（A口、B口和C口）以及一个控制寄存器。A_0、A_1地址信号经片内译码可产生4个有效地址，分别对应A、B、C这3个口和内部控制寄存器，具体规定如表7-4所示。

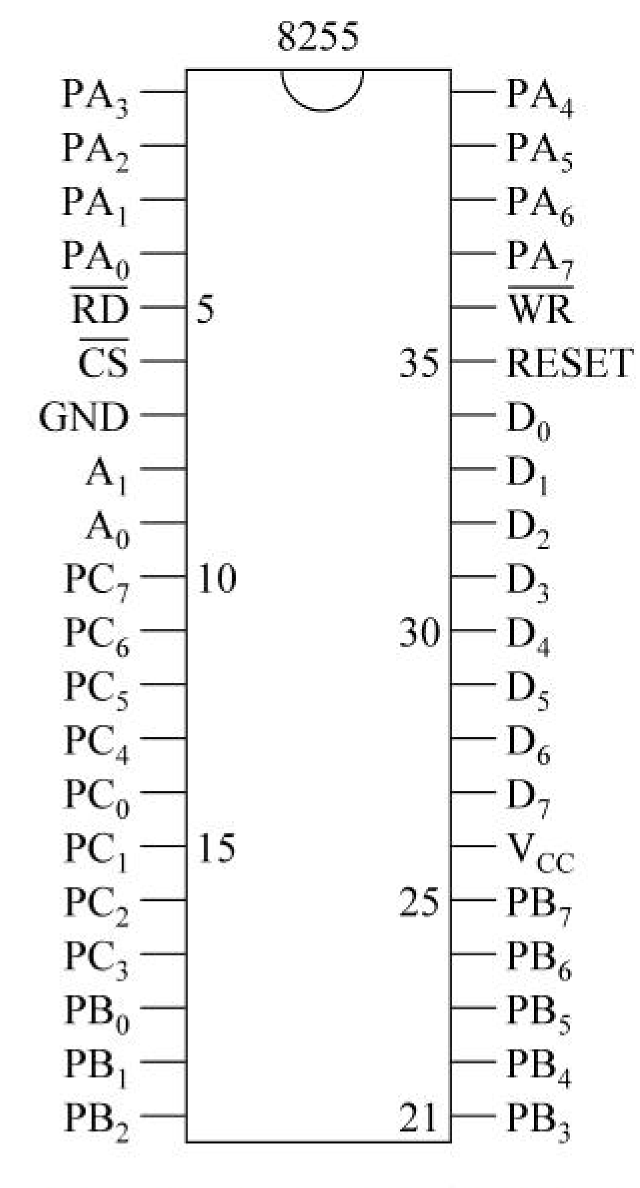


图7-18 8255的外部引线图

表7-4 各地址信号组合功能

A_1	A_0	选择	A_1	A_0	选择
0	0	A口	1	0	C口
0	1	B口	1	1	控制寄存器

在实际使用中，A_0、A_1通常接系统总线的A_0和A_1，它们与$\overline{CS}$一起来决定8255的接口地址。

(1) RESET：复位输入信号。通常接系统的复位 RESET 端。当它为高电平时使 8255 复位。复位后，8255 的 A 口、B 口和 C 口均被预设为输入状态。

(2) $PA_0 \sim PA_7$：A 口的 8 条输入输出信号线。这 8 条线是工作于输入、输出还是双向(同时为输入或输出)方式可由软件编程来决定。

(3) $PB_0 \sim PB_7$：B 口的 8 条输入输出信号线。利用软件编程可指定这 8 条线是作输入还是输出。

(4) $PC_0 \sim PC_7$：C 口的 8 条线，根据其工作方式可作为数据的输入或输出线，也可以用作控制信号的输出或状态信号的输入线，具体使用方法将在本节后面做介绍。

2. 内部结构

8255 的内部结构框图如图 7-19 所示，它由以下几个部分组成。

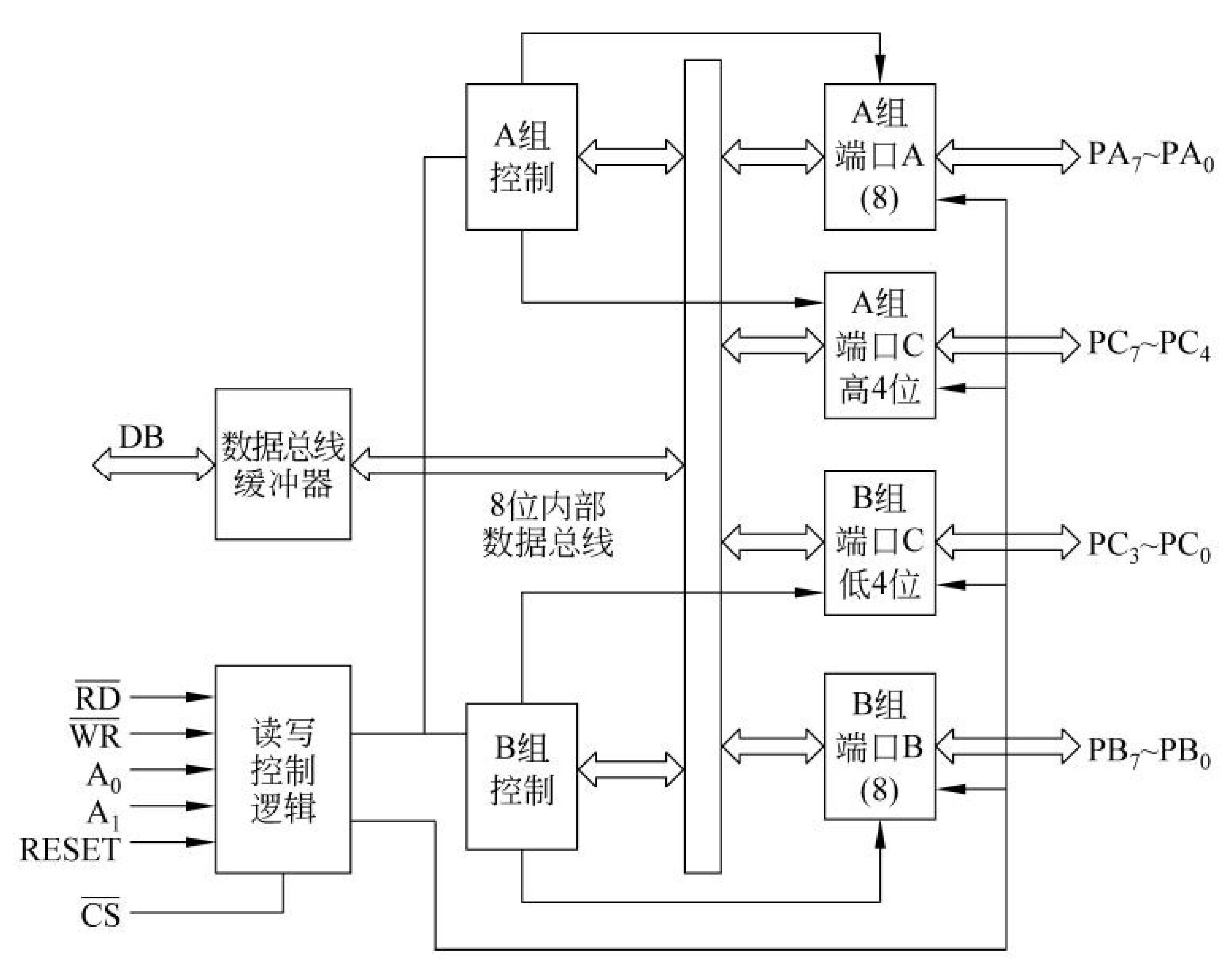


图 7-19 8255 内部结构框图

1) 数据端口

8255 有 A、B、C 共 3 个 8 位数据端口，可以通过编程把它们分别指定为输入口或输出口。A 口和 B 口的输入输出都具有数据锁存能力，C 口输出有锁存能力，而输入没有锁存能力。A、B、C 这 3 个口作输出时，其输出锁存器的内容可以由 CPU 用输入指令读回。在使用中，A、B、C 这 3 个口可作为三个独立的 8 位数据输入输出口；也可只将 A、B 口作为数据输入输出口，而使 C 口的各位作为它们与外设联络用的状态或选通控制信号的输入输出。C 口的主要特点是可以对其按位进行操作。

2) A 组和 B 组控制电路

从图 7-19 中可以看到，A 组和 B 组控制电路一方面接收读写控制逻辑电路的读写命令，另一方面接收由数据总线输入的控制字，分别控制 A 组和 B 组的读写操作和工作方式。A 组包括 A 端口的 8 位和 C 端口的高 4 位($PC_4 \sim PC_7$)，B 组包括 B 端口的 8 位和

C 端口的低 4 位($PC_0 \sim PC_3$)。编程写入的控制字输入到内部控制寄存器,控制 A 组和 B 组的工作方式。

3)读写控制逻辑

读写控制逻辑负责管理 8255 的数据传送。它接收来自系统总线的 A_0、A_1 和 $\overline{CS}$ 以及读($\overline{RR}$)、写($\overline{WR}$)和复位信号(RESET),并将这些信号进行逻辑组合,形成相应的控制命令,发送到 A 组和 B 组控制电路,以控制信息的传送。

4)数据总线缓冲器

数据总线缓冲器是一个三态双向 8 位数据缓冲器,8255 通过它和系统的数据总线相连,传递控制字、数据和状态信息。

图 7-20 给出了 8255 各引脚及端口在系统中的连接示意图。

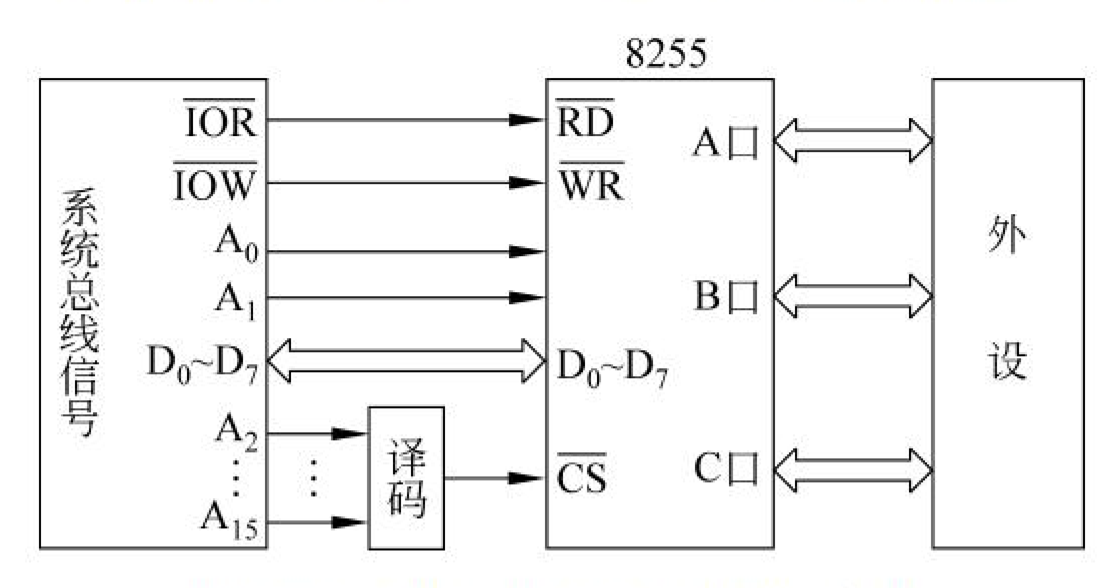


图 7-20 8255 与系统的连接示意图

7.3.2 8255 的工作方式

8255 有 3 种基本的工作方式:方式 0、方式 1 和方式 2。其中 A 口可以工作在方式 0、方式 1 和方式 2;B 口只能工作于方式 0 和方式 1;而 C 口在作为数据输入输出端口时,只能工作于方式 0。当 A、B 口工作在方式 1 或 A 端口工作于方式 2 时,C 口的某些位被用作连接相应的选通控制信号。3 个端口工作在哪一种方式下,可通过软件编程来设定。

1. 工作方式 0

方式 0 又称为基本输入输出方式。方式 0 的示意图如图 7-21 所示。

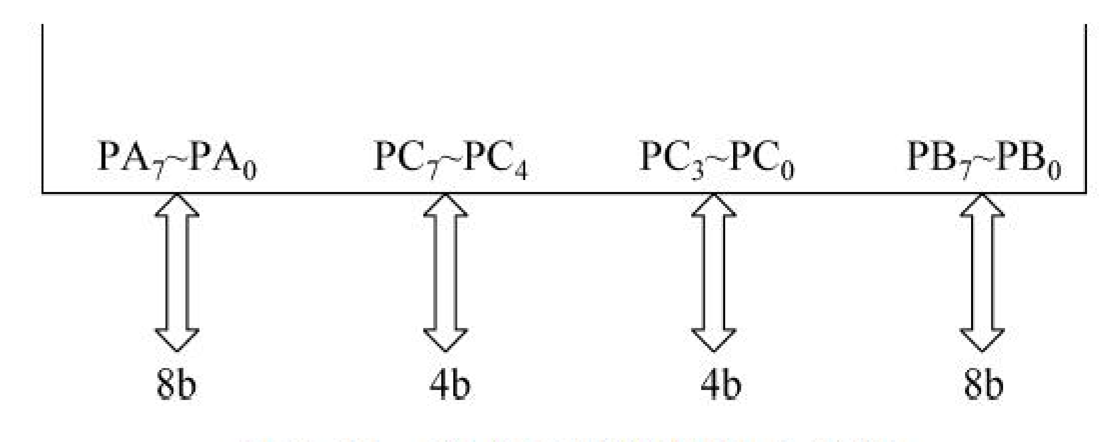


图 7-21 方式 0 下的端口示意图

这种方式有如下两点特点。

(1) A 口、C 口的高 4 位,B 口以及 C 口的低 4 位可分别定义为输入或输出,各端口

互相独立，故共有 16 种不同的组合。例如可定义 A 口和 C 口高 4 位为输入口，B 口和 C 口低 4 位为输出口；或 A 口为输入，B 口、C 口高 4 位、C 口低 4 位为输出；等等。

(2) 在方式 0 下，C 口有按位进行置位和复位的能力。有关 C 口的按位操作见后续内容。

方式 0 最适合用于无条件传送方式，由于传送数据的双方互相了解对方，所以既不需要发控制信号给对方，也不需要查询对方状态，故 CPU 只需直接执行输入输出指令便可将数据读入或写出。

方式 0 也能用于查询工作方式，由于没有规定固定的应答信号，这时常将 C 口的高 4 位(或低 4 位)定义为输入口，用来接收外设的状态信号；而将 C 口的另外 4 位定义为输出口，输出控制信息。此时的 A、B 口可用来传送数据。

2. 工作方式 1

方式 1 也称为选通输入输出方式。在这种方式下，A 口和 B 口仍作为数据的输出口或输入口，但数据的输入输出要在选通信号控制下来完成。这些选通信号利用 C 口的某些位来提供。A 口和 B 口可独立地由程序任意指定为数据的输入口或输出口。为方便起见，下面分别以 A 口、B 口均作为输入或均作为输出来加以说明。

1) 方式 1 下 A 口、B 口均为输出

此时要利用 C 口的 6 条线作为选通控制信号线，其定义如图 7-22 所示。所用到的 C 口的信号线是固定不变的，A 口使用 PC_3、PC_6 和 PC_7，而 B 口用 PC_0、PC_1 和 PC_2。方式 1 下数据的输出过程如下。

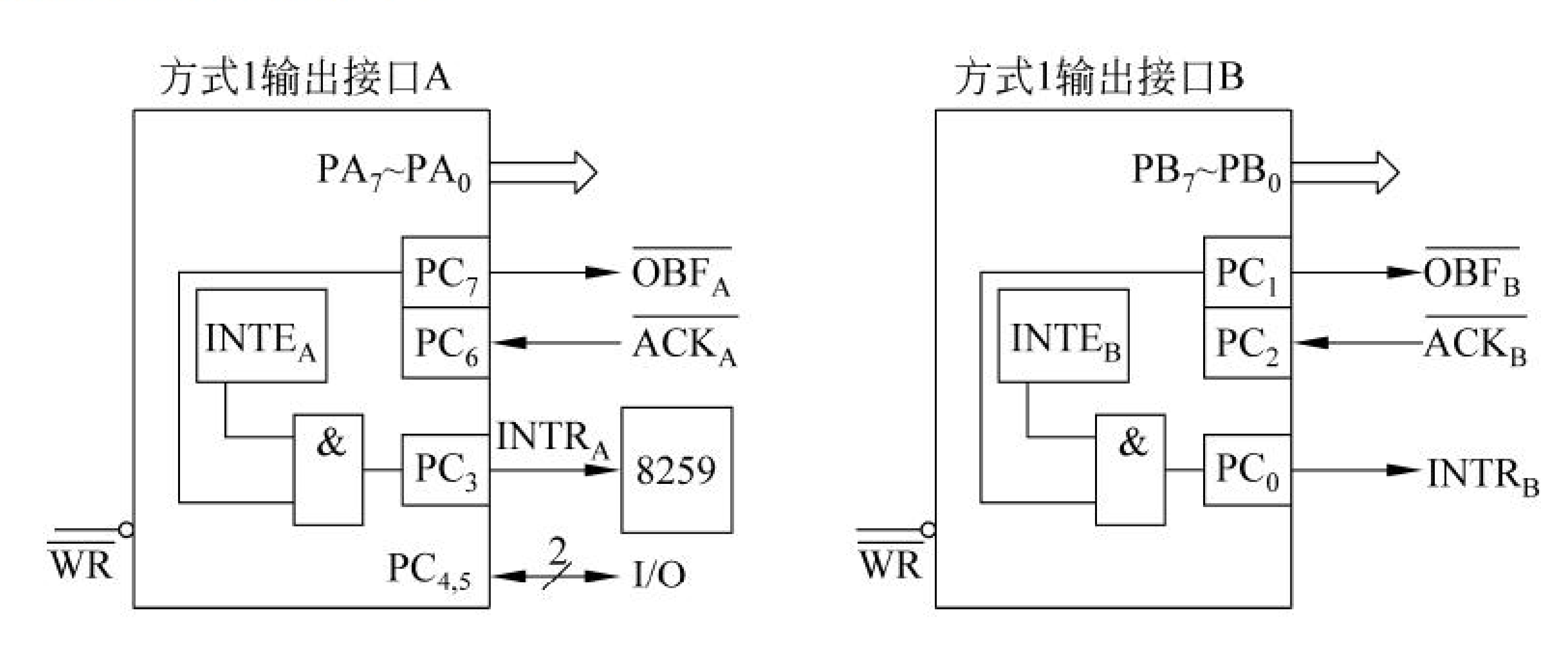


图 7-22　方式 1 下 A、B 口为输出的选通信号定义

(1) 系统在 IOW 信号有效期间将数据输入到 A 端口或 B 端口。

(2) 接口输出缓冲器满信号$\overline{OBF}$(低电平有效)通知外设，在规定的端口上已有一个有效数据，外设可以从该端口读走数据。

(3) 外设从该端口取走数据后，发出响应信号$\overline{ACK}$(低电平有效)，同时使$\overline{OBF}$=1。

(4) 外设取走一个数据后，其$\overline{ACK}$信号的上升沿产生有效的 INTR 信号，该信号用于通知 CPU 可以再输出下一个数据。INTR 的有效条件为$\overline{OBF}$=1，$\overline{ACK}$=1，INTE=1。

(5) 8255 内部有一个内部中断触发器，如图 7-22 所示，当中断允许状态 INTE 为高电平，且$\overline{OBF}$也变高时，产生有效的 INTR 信号。INTE 由 PC_6(端口 A)或 PC_2(端口 B)的置位/复位控制。

INTE 是否输出高电平由$\overline{ACK}$信号决定。以 A 口为例，当 CPU 向接口写数据时(执行一条 OUT 指令)，在$\overline{IOW}$有效期间将数据锁存于芯片的数据缓冲器中，之后在$\overline{IOW}$的上升沿使$\overline{OBF}=0$(PC_7 端输出负脉冲)，通知外部设备 A 口已有数据准备好。一旦外设将数据接收，就送出一个有效的$\overline{ACK}$脉冲，该脉冲使$\overline{OBF}=1$，同时使 INTE 也为高电平，从而在 PC_3 端产生一个有效的 INTR 信号。该信号可接到中断控制器 8259 的 IR 端，进而向 CPU 提出中断请求。CPU 响应中断后，向接口写入下一个数据，同样由$\overline{IOW}$将数据锁存，当数据锁存并由信号线输出时，8255 就去掉 INTR 信号并使$\overline{OBF}$有效，重复上述过程。方式 1 下的整个输出过程也可参考图 7-23 所示的简单时序。

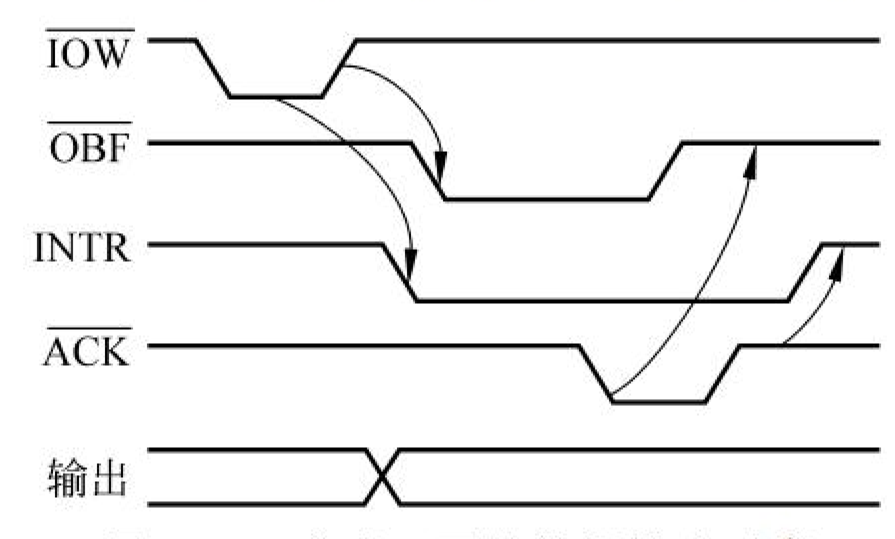


图 7-23　方式 1 下的数据输出时序

当 A 口和 B 口同时工作于方式 1 输出时，仅使用了 C 口的 6 条线，剩余的两位可以工作于方式 0，实现数据的输入或输出，其数据的传送方向可由程序指定；也可通过位操作方式对它们进行置位或复位。当 A、B 两个口中仅有一个口工作在方式 1 时，只用去 C 口 3 条线，则剩下的 5 条线也可按照上面所说的方式工作。

2）方式 1 下 A 口、B 口均为输入

与方式 1 下两端口均为输出类似，要实现选通输入，同样要利用 C 口的信号线。其定义如图 7-24 所示。A 口使用了 C 口的 PC_3、PC_4 和 PC_5，B 口同样用了 PC_0、PC_1 和 PC_2。

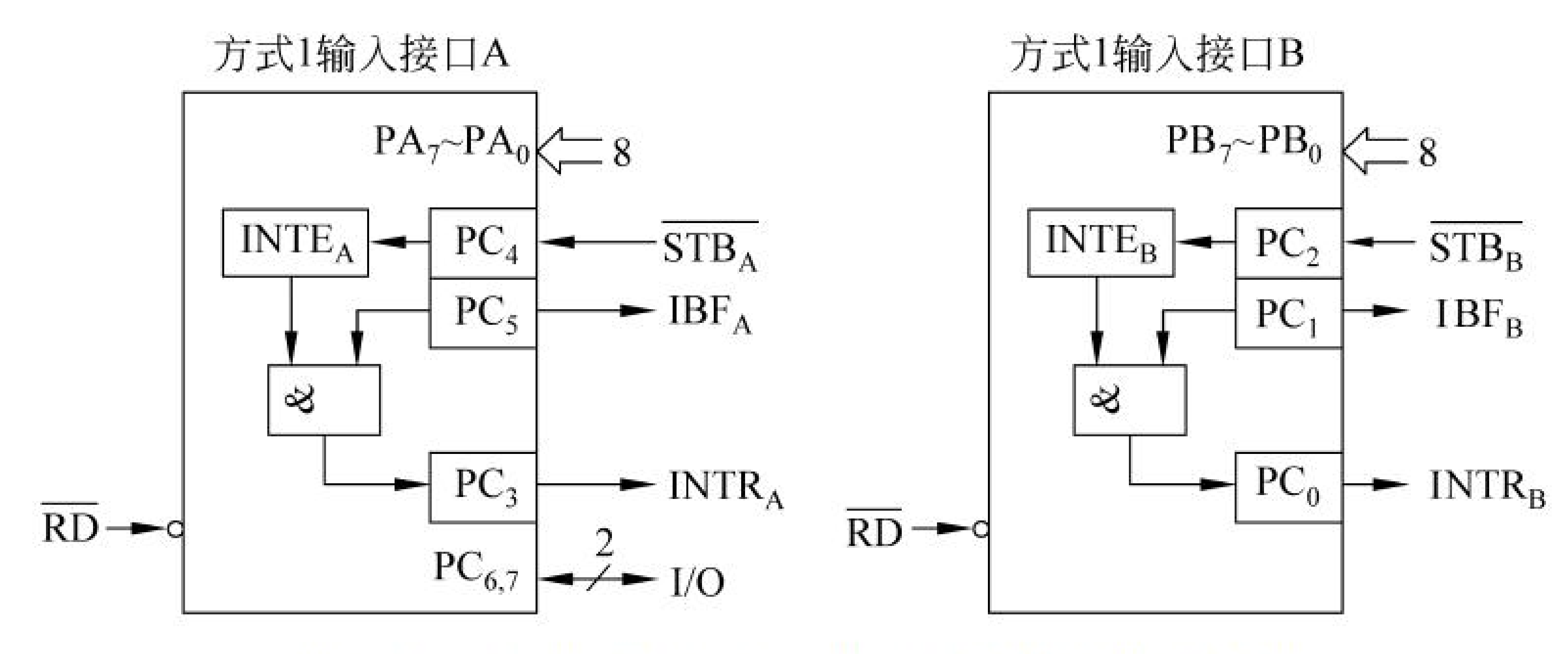


图 7-24　方式 1 下 A、B 口均为输入时的信号定义

方式 1 下数据的输入过程可描述如下。

(1) 外部设备发出低电平有效的$\overline{STB}$信号，并在$\overline{STB}$有效期间将数据锁存于 8255 的输入数据缓冲器中。

(2) 当输入缓冲器满后，接口发出高电平有效的 IBF 信号。它作为$\overline{STB}$的应答信号，表示 8255 的缓冲器中有一个数据尚未被 CPU 读走。外设可使用此信号来决定是否能送下一个数据。

(3) 当$\overline{STB}=1$时会使内部中断触发器 INTE 和 IBF 均为高电平，产生有效的 INTR 信号，向 CPU 提出中断请求。

(4) INTR 信号可用于通过 8259 向 CPU 提出中断请求，要求 CPU 从 8255 的端口上

读取数据。CPU 响应中断并读取数据后使 IBF 和 INTR 变为无效。上述过程可用图 7-25 的简单时序图进一步说明。

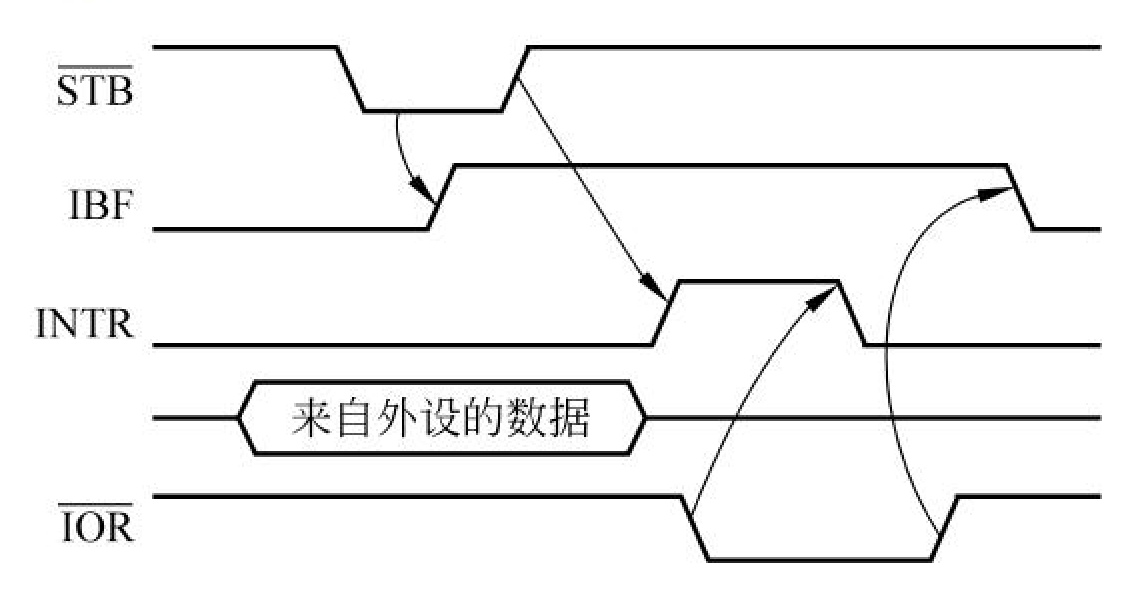


图 7-25　方式 1 下数据输入时序图

在方式 1 下输入数据时，INTR 同样受中断允许状态 INTE 的控制。INTE 的状态可利用 C 口位操作方式的置位/复位来控制。例如，用位操作方式使 $PC_4=1$，则 A 口的 $INTE_A$ 为 1，允许中断；使 $PC_4=0$ 则禁止中断。B 口的 $INTE_B$ 是由 PC_2 控制的。

在方式 1 之下，8255 的 A 口和 B 口既可以同时为输入或输出，也可以一个为输入，另一个为输出；还可以使这两个端口一个工作于方式 1，而另一个工作于方式 0。这种灵活的工作特点是由其可编程的功能决定的。

3. 工作方式 2

方式 2 又称为双向传输方式。只有 A 口可以工作在这种方式下。双向方式使外设能利用 8 位数据线与 CPU 进行双向通信，既能发送数据，也能接收数据。即此时 A 口既作为输入口又作为输出口。与方式 1 类似，方式 2 要利用 C 口的 5 条线来提供双向传输所需的控制信号。当 A 口工作于方式 2 时，B 口可以工作在方式 0 或方式 1，而 C 口剩下的 3 条线可作为输入输出线使用或用作 B 口方式 1 之下的控制线。

A 口工作于方式 2 下时的各信号定义如图 7-26 所示。图中省略了 B 口和 C 口的其他引线。当 A 口工作于方式 2 时，其控制信号 $\overline{OBF}$、$\overline{ACK}$、$\overline{STB}$、IBF 及 INTR 的含义与方式 1 时相同。但在时序上有一些不同，主要如下。

(1) 因为在方式 2 下，A 口既作为输出又作为输入，因此，只有当 $\overline{ACK}$ 有效时，才能打开 A 口输出数据三态门，使数据由 PA_0～PA_7 输出；当 $\overline{ACK}$ 无效时，A 口的输出数据三态门呈高阻状态。

(2) 此时 A 口输入、输出均有数据的锁存能力。

(3) 方式 2 下，A 口的数据输入或数据输出均可引起中断。由图 7-26 可见，输入或输出中断还受到中断允许状态 $INTE_2$ 和 $INTE_1$ 的影响。$INTE_2$ 是由 PC_4 控制的，而 $INTE_1$ 是由 PC_6 控制的。利用 C 口的按位操作，使 PC_4 或 PC_6 置位或复位，可以允

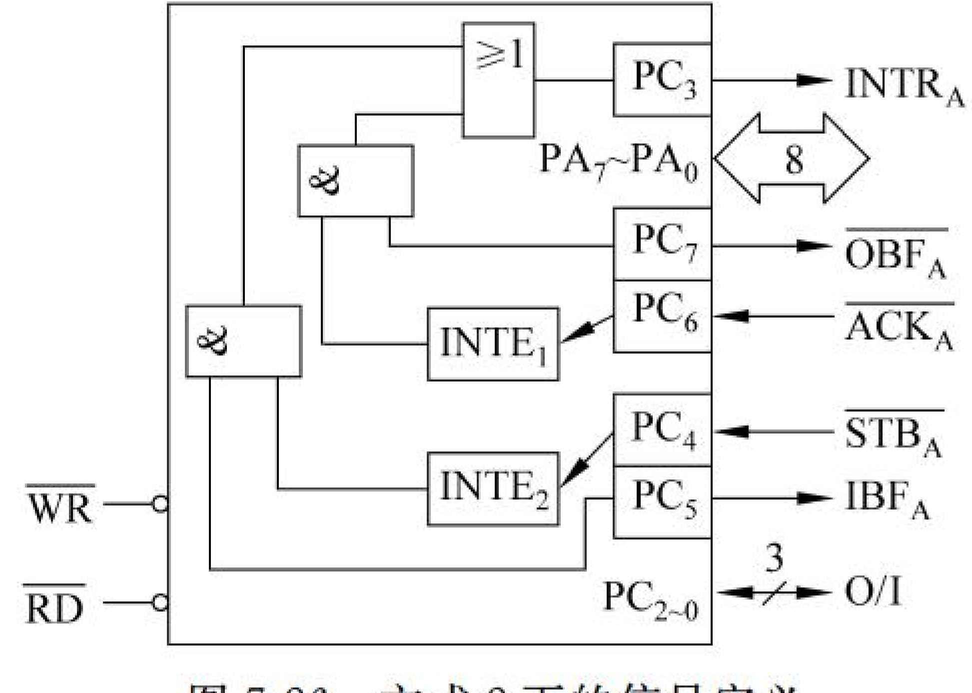


图 7-26　方式 2 下的信号定义

许或禁止相应的中断请求。

A 口工作于方式 2 的时序如图 7-27 所示。此时的 A 口可以认为是前面方式 1 的输入和输出相结合而分时工作。实际传输过程中，输入和输出的顺序以及各自操作的次数是任意的，只要$\overline{\text{IOW}}$在$\overline{\text{ACK}}$之前发出、$\overline{\text{STB}}$在$\overline{\text{IOR}}$之前发出就可以了。

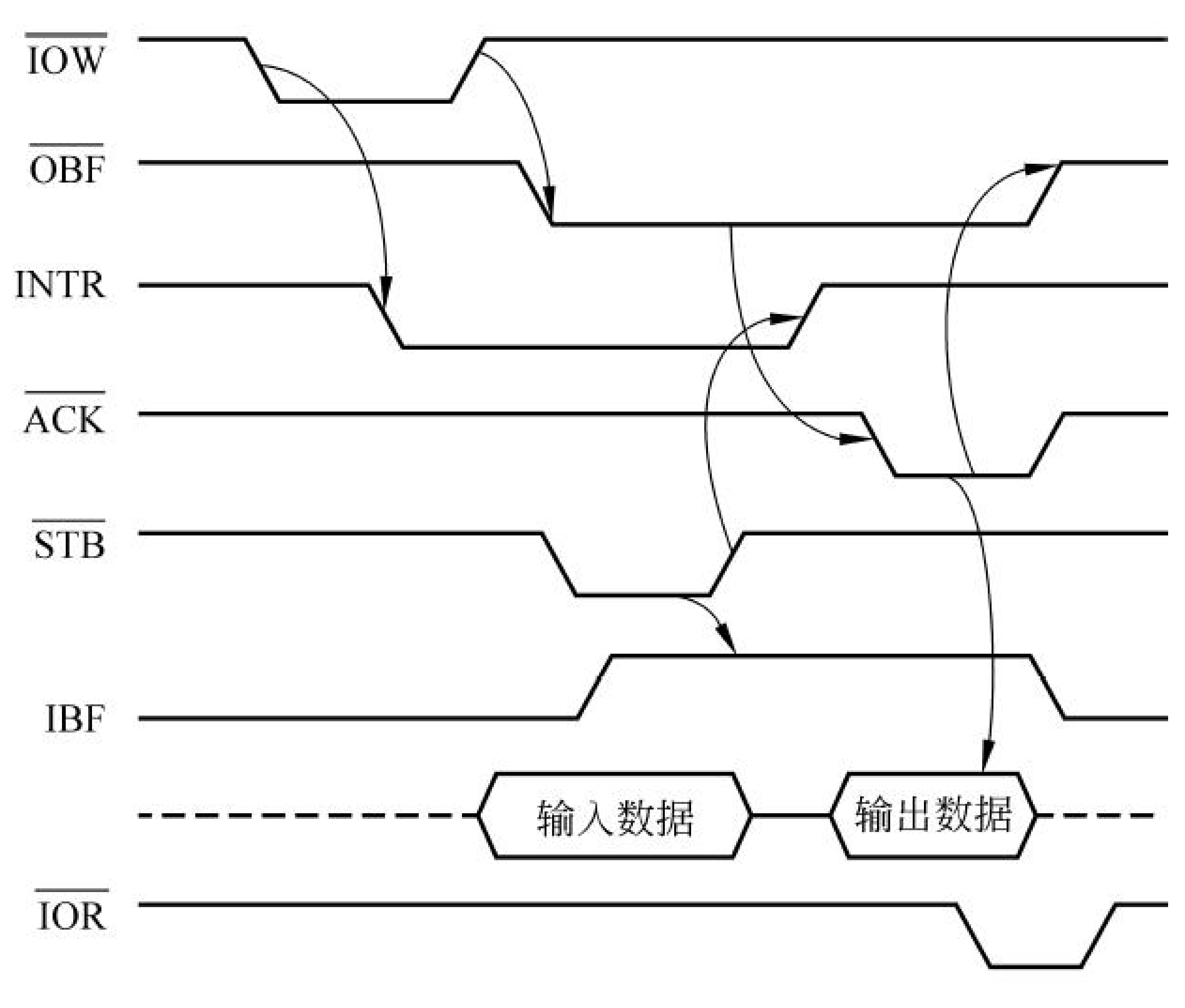


图 7-27 方式 2 下的工作时序

在输出时，CPU 发出写脉冲$\overline{\text{IOW}}$，向 A 口写入数据；$\overline{\text{IOW}}$信号使 INTR 变低电平，同时使$\overline{\text{OBF}}$有效；外设接到$\overline{\text{OBF}}$信号后发出$\overline{\text{ACK}}$信号，从 A 口读出数据；$\overline{\text{ACK}}$信号使$\overline{\text{OBF}}$无效，并使 INTR 变高，产生中断请求，准备输出下一个数据。

输入时，外设向 8255 送来数据，同时发$\overline{\text{STB}}$信号给 8255，该信号将数据锁存到 8255 的 A 口，从而使 IBF 有效；$\overline{\text{STB}}$信号结束使 INTR 有效，向 CPU 请求中断；CPU 响应中断后，发出读信号$\overline{\text{IOR}}$，从 A 口中将数据读走；$\overline{\text{IOR}}$信号会使 INTR 和 IBF 信号无效，从而开始下一个数据的读入过程。

在方式 2 下，8255 的 PA_0～PA_7 引线上随时可能出现输出到外设的数据，也可能出现外设送给 8255 的数据，这需要防止 CPU 和外设同时竞争 PA_0～PA_7 数据线的问题。

7.3.3 8255 的控制字及状态字

由前面的叙述已知，8255 具有 3 种工作方式，可以利用软件编程来指定 8255 的 3 个端口当前工作于何种方式。这里所谓的软件编程，就是向芯片中的控制寄存器送入不同的控制字，从而确定 8255 的工作方式。这种通过软件来确定 8255 工作方式的过程称为 8255 的初始化，在实际应用中，可根据不同的需要，通过初始化使 8255 的 3 个端口工作在不同的方式(当然，B 口只能工作于方式 0 和方式 1，而 C 口只能工作于方式 0)。

1. 控制字

8255 的控制字包括用于设定 3 个端口工作方式的方式控制字，如图 7-28(a)所示，以

及用于将C口某一位初始化为某个确定状态（“0”或“1”）的位控制字，如图7-28(b)所示。两个控制字均由8位二进制数组成，由最高位(D_7)的状态决定当前的控制字是方式控制字还是C口的按位操作控制字。控制字各位的含义如图7-28所示。

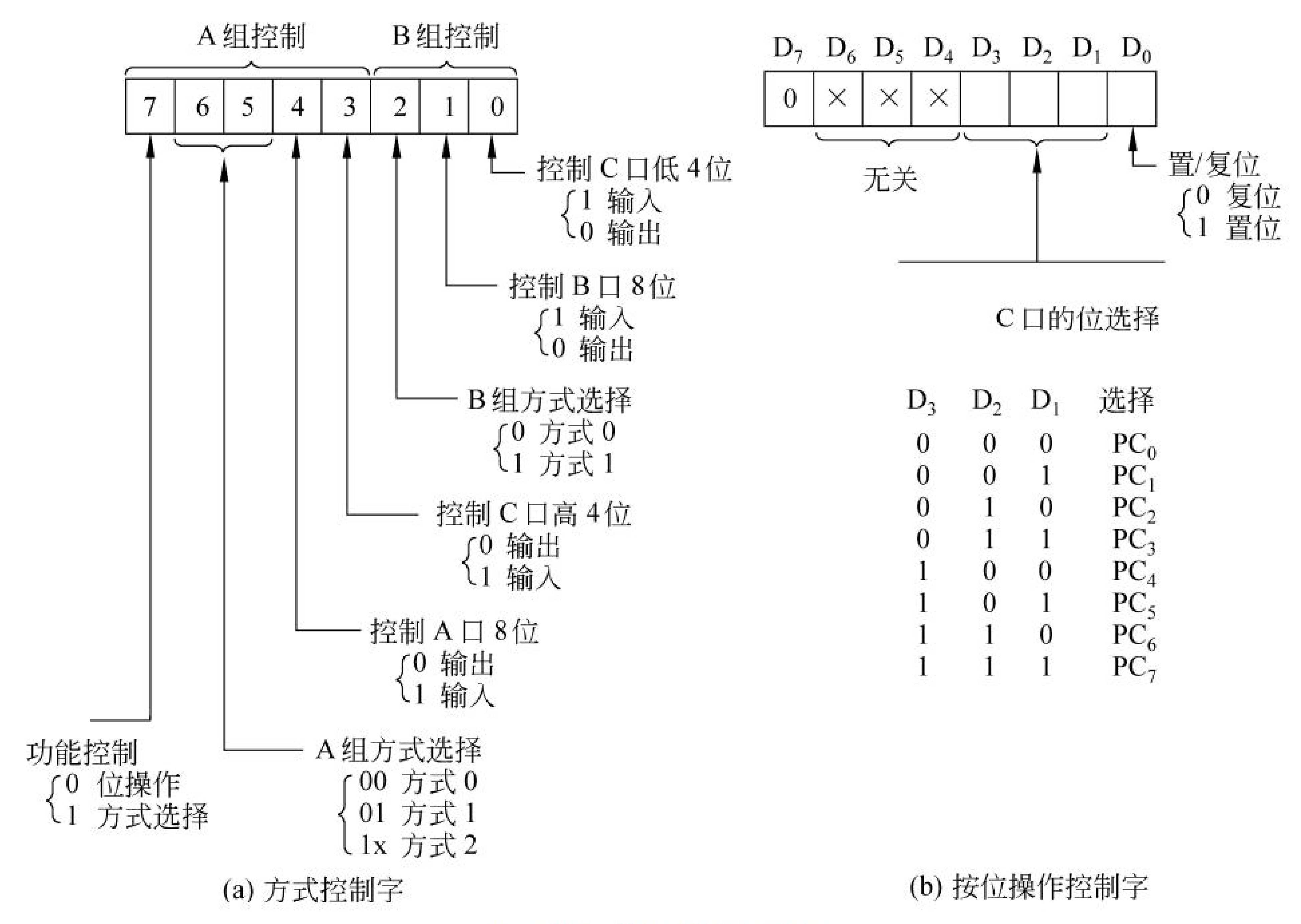


图7-28　8255的控制字

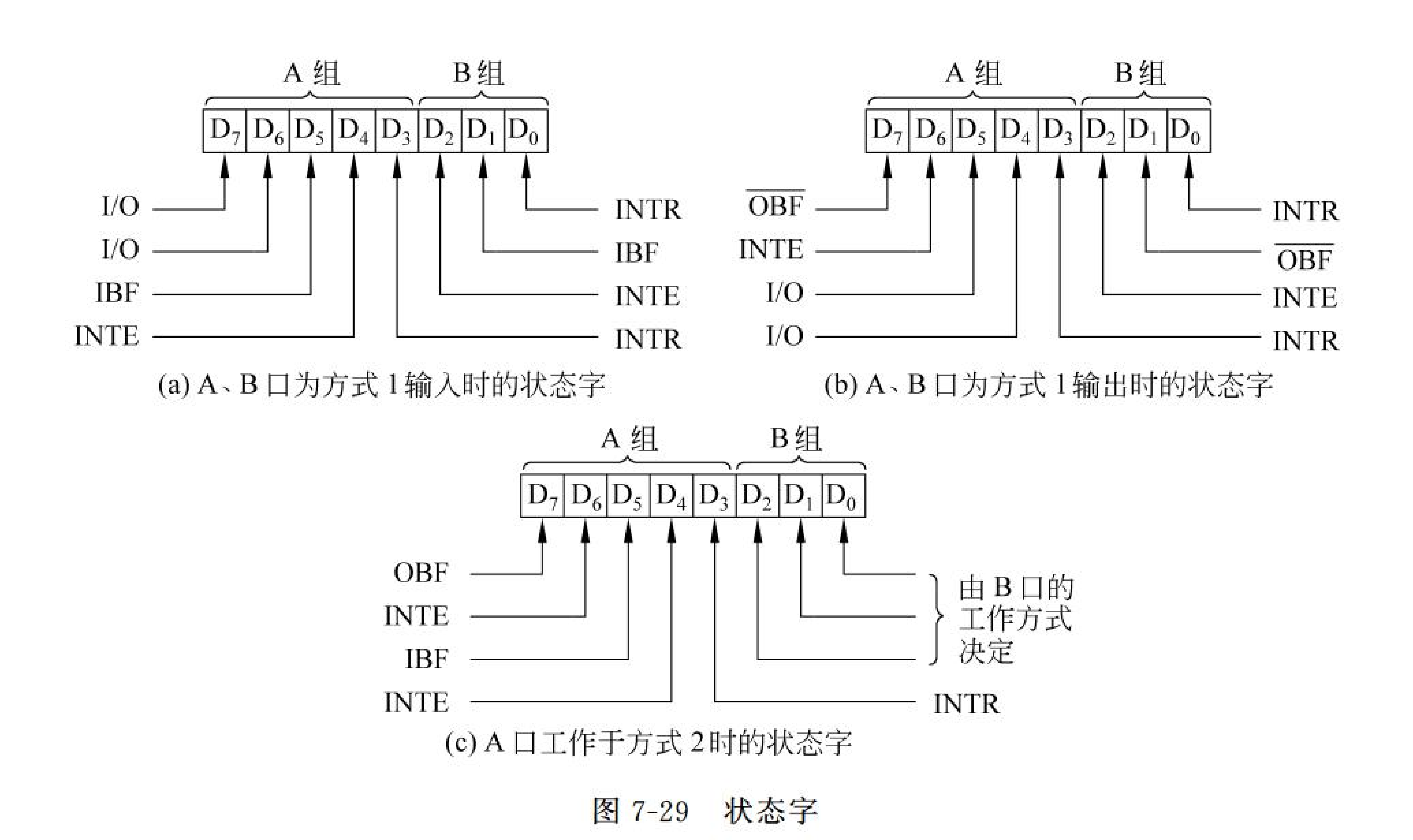
由图可知，当$D_7=1$时，该控制字为方式控制字，用于确定各端口的工作状态。$D_6 \sim D_3$用来控制A组，即A口的8位和C口的高4位；控制字的低3位$D_2 \sim D_0$用来控制B组，包括B口的8位和C口的低4位。

当$D_7=0$时，指定该控制字为对C口进行位操作控制——按位置位或复位。在必要时，可利用该控制字使C口的某一位输出0或1。

2. 状态字

状态字反映了C端口各位当前的状态。当8255的A口、B口工作在方式1或A口工作在方式2时，通过读C口的状态可以检测A口和B口当前的工作情况。A、B口工作在不同方式下的状态字各位的含义分别如图7-29(a)、(b)和(c)所示，其中低3位$D_0 \sim D_2$由B口的工作方式来决定。当为方式1输入时，其定义如图7-29(a)；当工作在方式1输出时，与图7-29(b)所定义的$D_0 \sim D_2$相同。

需要说明的是，图7-29(a)和(b)分别表示在方式1之下，A口、B口同为输入或同为输出的情况。若在此方式下，A口、B口各为输入或输出时，状态字为上述两状态字的组合。



(a) A、B口为方式1输入时的状态字

(b) A、B口为方式1输出时的状态字

(c) A口工作于方式2时的状态字

图 7-29 状态字

7.3.4 8255的应用

1. 8255与系统的连接

8255内部包括A、B、C共3个端口和一个控制寄存器，共占4个外设地址。由高位地址通过译码产生片选信号，决定芯片在整个接口地址空间中的位置；A_1、A_0决定片内的4个端口（例如，$A_1A_0=00$时指向的是A口），它们结合起来共同决定了芯片所占的地址范围。

对8255内部的每一个端口都可以分别进行读写操作。例如，读A口是CPU将A口的数据读入AL寄存器；写A口是CPU将AL中的数据写入A口输出。对这4个地址进行不同操作时各引脚的状态如表7-5所示。根据该表，可以很方便地实现8255与系统总线的连接。

表 7-5 8255各引脚状态

CS	A_1	A_0	IOR	IOW	操　作
0	0	0	0	1	读A口
0	0	1	0	1	读B口
0	1	0	0	1	读C口
0	0	0	1	0	写A口
0	0	1	1	0	写B口
0	1	0	1	0	写C口
0	1	1	1	0	写控制寄存器
1	×	×	1	1	D_0～D_7三态

图 7-20 曾给出了芯片与系统的连接框图。在该图中,数据信号线、读写控制信号线以及片内地址信号 A_0、A_1 都与系统相应信号线直接相连,3 个端口的位数据线根据具体的应用连接到相应外部设备。因此,8255 芯片与系统连接线路的设计主要在译码电路上。

图 7-30 所示的是利用全译码方式将一片 8255 连接到系统总线上的连接示例。图中芯片所占的地址范围由 A_{15}～A_2 决定,为 0FF00H～0FF03H。而 A_0 和 A_1 的状态则决定寻址芯片的哪一个端口或控制寄存器。

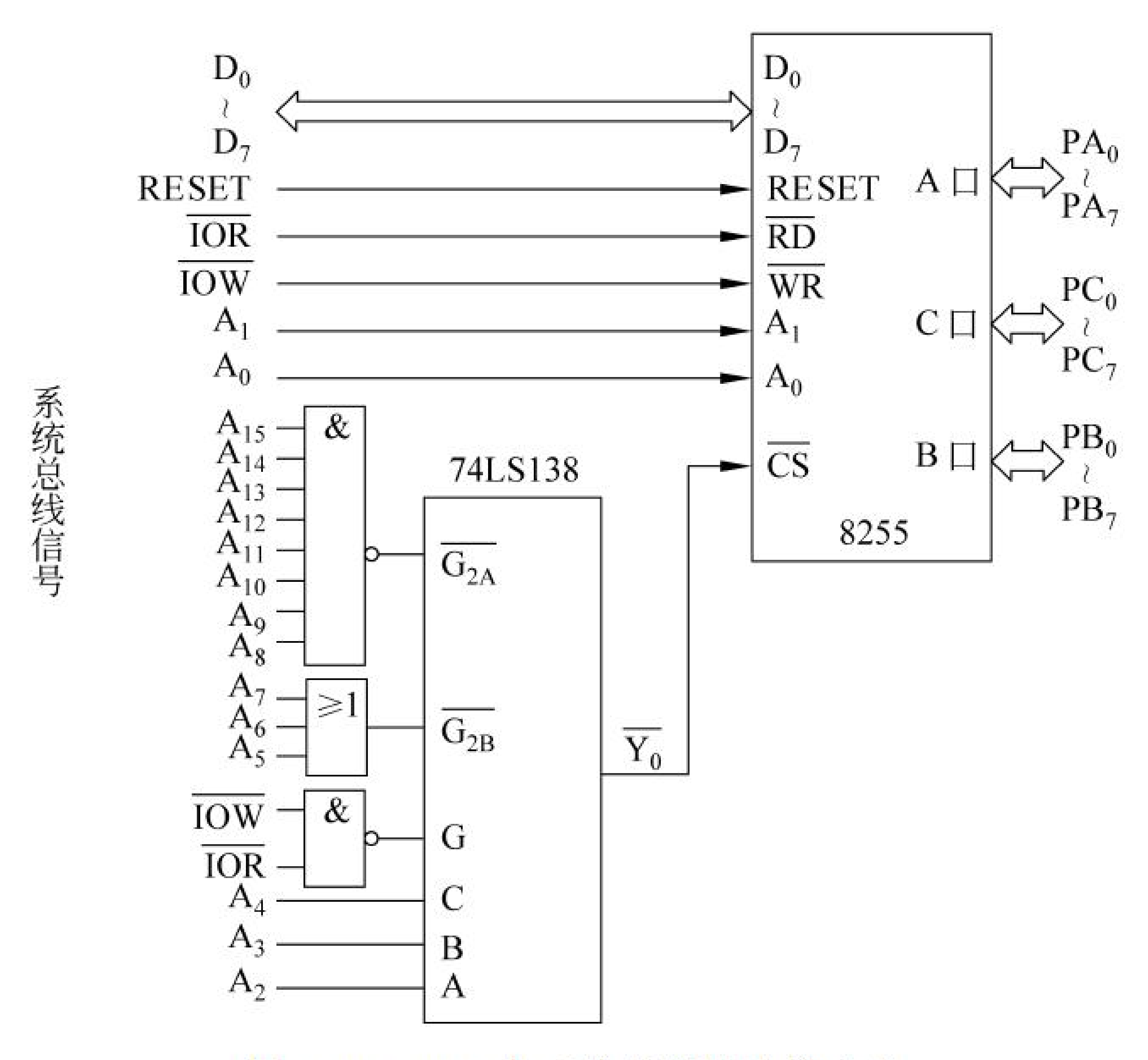


图 7-30 8255 与系统总线的连接方法

2. 软件设计

在硬件线路设计完成后,需要进行相应的软件设计。对于 8255 可编程接口,软件设计包括初始化程序设计和实现数据传输的控制程序设计两部分。

作为可编程接口,8255 在使用时首先需要初始化,即将适当的控制字写入 8255 的控制寄存器中。只有在初始化结束后,才能够进行正常的数据传送。在数据传送过程中,CPU 还要通过 8255 向外设发出控制信号并接收外设的状态信息。数据传送的方式可根据外部设备的性质及具体的应用,采用第 6 章所介绍的各种输入输出方法。

3. 应用实例

下面通过应用实例来进一步说明 8255 的应用。

【例 7-3】 利用 8255 作为打印机的连接接口,打印机的工作时序如图 7-31 所示,通过该打印机接口打印字符串,字符串长度放在数据段的 COUNT 单元中,要打印的字符存放在从 DATA 开始的数据区中。

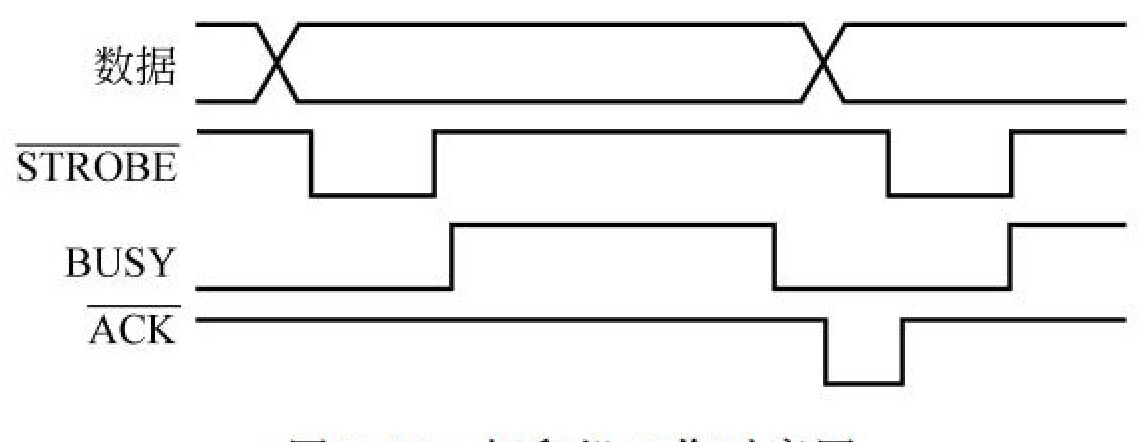


图 7-31　打印机工作时序图

要求 8255 芯片的地址范围为 FBC0H～FBC3H。

题目分析：

由图 7-31 可知，数据锁存信号$\overline{STROBE}$在初始时为高电平，当系统通过 8255 接口将打印的字符送到打印机的 D_0～D_7 端时，应紧接着送出低电平的$\overline{STROBE}$信号（宽度≥1μs），将数据锁存在打印机内部，以便处理。同时，打印机的 BUSY 端送出高电平信号，表示其正忙。仅当 BUSY 端信号变低后，CPU 才可以将下一个数据送给打印机。

实现数据的打印输出既可以采用查询工作方式，也可以采用中断控制方式。根据上述需求，例中采用查询工作方式，即：使 8255 工作于方式 0。作为数据输出的端口既可以是图 7-32 中的 A 口，也可以选用 B 口。考虑到 C 端口可以分为两个 4 位端口的特点，通常用 C 端口来连接控制或状态信号。

设计 8255 与系统及打印机的连接如图 7-32 所示。选用 A 口作为数据输出，向打印机输出输出数据；利用 C 口的 PC_6 输出$\overline{STROBE}$锁存信号，在低 4 位中选取 PC_1 作为 BUSY 信号的输入。B 端口不使用，初始化时可任意定义为输入或输出。

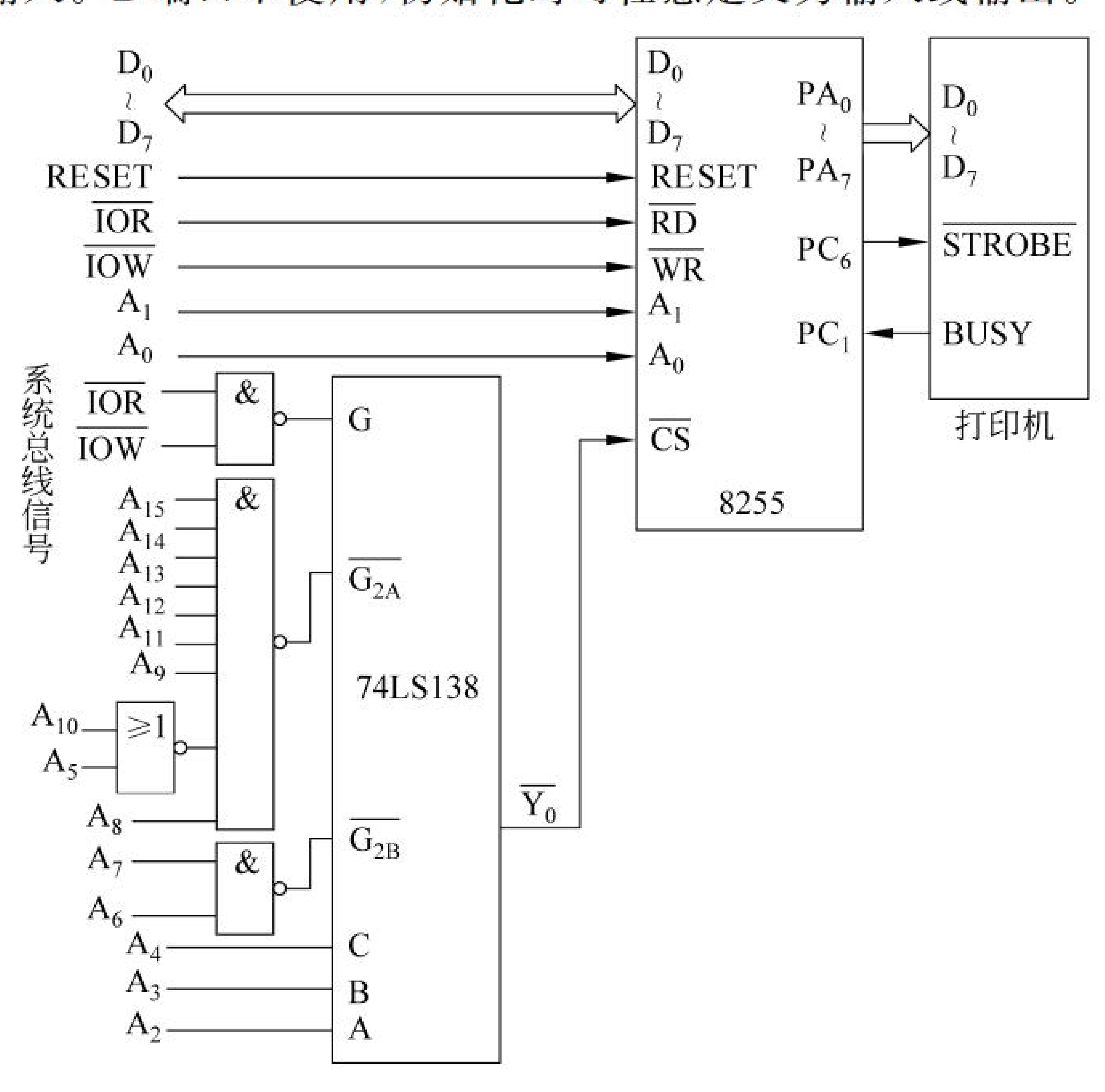


图 7-32　8255 与打印机的连接

8255的初始化程序如下(由于数据输出后要通过 PC_6 端输出一个负脉冲,故在初始化时先要将 PC_6 初始化为高电平):

```
INIT: MOV  DX, 0FBC3H          ;8255的控制寄存器端口地址送 DX
      MOV  AL, 10000001B       ;A组方式 0,A口输出,C口高 4位输出
                               ;B组方式 0,B口输出,C口低 4位输入
      OUT  DX, AL              ;方式控制字送控制寄存器
      MOV  AL, 00001101B       ;C口的按位操作控制字,使 PC6 初始状态置为 1
      OUT  DX, AL              ;C口位操作控制字送控制寄存器
```

下面是打印一批字符的程序段:

```
       MOV   CX, COUNT         ;将字符串长度作为循环次数
       MOV   SI, OFFSET DATA   ;取字符串首地址
GOON:  MOV   DX, 0FBC2H        ;0FBC2H为 C口的地址
       IN    AL, DX            ;从 C口读入打印机的 BUSY信号状态
       AND   AL, 02H
       JNZ   GOON              ;若 BUSY为高电平则循环等待
       MOV   AL, [SI]          ;否则取一个字符
       MOV   DX, 0FBC0H        ;0FBC0H为 A口的地址
       OUT   DX, AL            ;输出一个字符到 A口
       MOV   DX, 0FBC2H        ;准备在 PC6 上生成一个负脉冲
       MOV   AL, 0
       OUT   DX, AL            ;因仅 PC6 接打印机,故由 C口输出 00H将使 PC6 变低
       MOV   AL, 40H
       OUT   DX, AL            ;再使 PC6 变高,在 PC6 上生成一个STROBE负脉冲
       INC   SI                ;指向下一个字符
       LOOP  GOON              ;若未结束则继续
       HLT
```

在上面程序中,$\overline{\text{STROBE}}$负脉冲是通过往 C 口输出数据(先将 PC_6 初始化为 1,然后输出一个 0,再输出一个 1)而形成的。当然,也可以利用控制字对 C 口的按位置位/复位操作来实现。例如:

```
MOV  DX, 0FBC3H
MOV  AL, 00001100B          ;PC6 复位(=0)
OUT  DX, AL
MOV  AL, 00001101B          ;PC6 置位(=1)
OUT  DX, AL
```

【例 7-4】 对例 7-3,利用中断控制方式实现数据的打印输出。

题目分析:

若采用中断控制方式实现数据传送,则应使 8255 工作在方式 1 下。从图 7-31 所示的打印机工作时序可知,打印机每接收一个字符后,会送出一个低电平的响应信号$\overline{\text{ACK}}$。利用这个信号,可使工作于方式 1 的 8255 通过中断来打印字符。

设置 8255 芯片的 A 端口为数据输出口，此时 PC_7 自动作为 $\overline{OBF}$ 信号的输出端，PC_6 自动作为 $\overline{ACK}$ 信号的输入端，而 PC_3 则自动作为 INTR 信号的输出端，将其接到 8259 的 IR_2 端，所以中断类型号为 0AH。

要使 PC_3 能够产生中断请求信号 INTR，还必须使 A 口的中断请求允许状态 INTE=1。这是通过 8255 的置位/复位操作将 PC_6 置 1 来实现的(参见方式 1 下的数据输出时序图 7-23)，即在初始化 8255 时除写方式控制字外，还要写 C 口的位操作控制字。

输出时，先输出一个空字符，以引起中断过程。在中断中输出要打印的字符，利用 $\overline{OBF}$ 的下降沿触发一单稳触发器，产生打印机所需要的 $\overline{STROBE}$ 脉冲，将字符锁存到打印机中。接收到字符后，打印机发出 $\overline{ACK}$，清除 $\overline{OBF}$ 标志并产生有效的 INTR 输出，形成新的中断请求，CPU 响应中断后再输出下一个字符。

为简单起见，在初始化 8255 时，仍使 B 口工作于方式 0 输出，C 口的其余 5 条线均定义为输出，故控制字为 10100000B，即 0A0H。

设计 8255 与打印机的电路连接方法如图 7-33 所示。

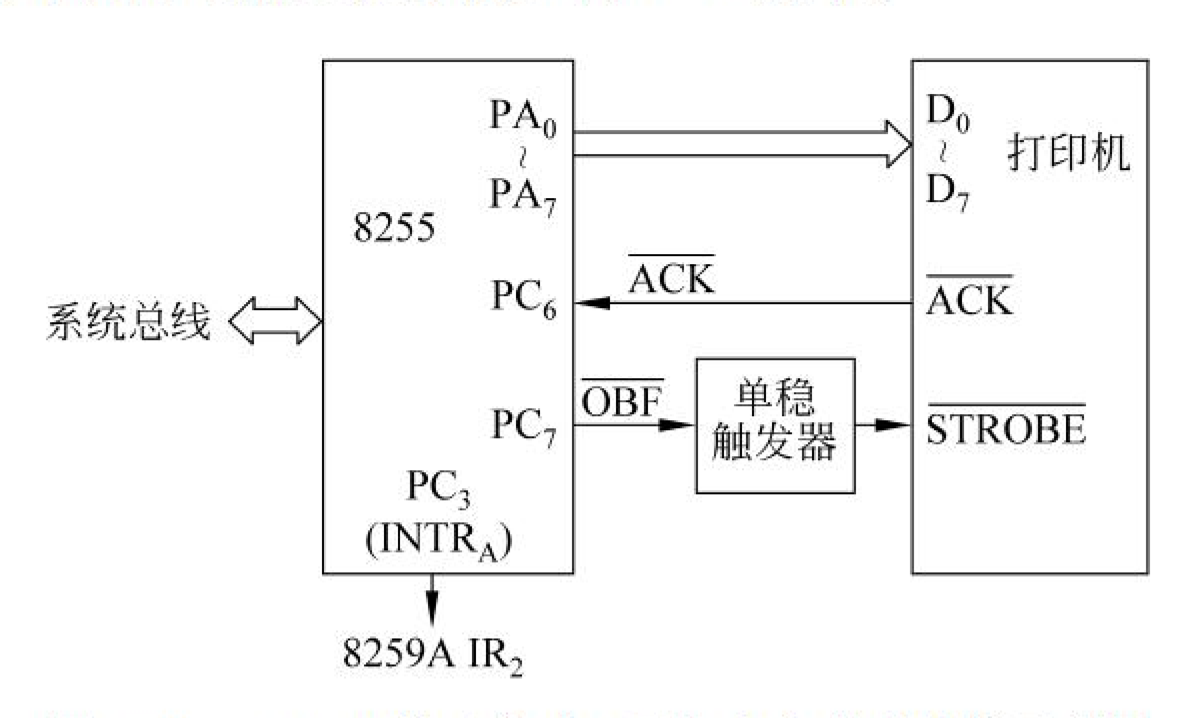


图 7-33　8255 工作于方式 1 下与打印机的连接示意图

以下是向打印机输出字符的程序，包括主程序和中断服务程序两部分。主程序完成以下 3 项工作：将中断服务子程序的入口地址送中断向量表、开中断等中断的准备工作以及 8255 的初始化。而中断服务子程序则完成字符的输出。C 程序中假设 8259A 的端口地址为 0FF00H(A_0=0)和 0FF01H(A_0=1)。

```
MAIN: PUSH  DS
      LEA   DX, PRINT
      MOV   AX, SEG PRINT
      MOV   DS, AX
      MOV   AL, 0AH
      MOV   AH, 25H
      INT   21H
      POP   DS                      ;设置中断向量
      MOV   DX, 0FBC3H
      MOV   AL, 0A0H                ;8255初始化：A口方式 1,输出,B口方式 0,输出,
      OUT   DX, AL                  ;C口其余的 5条线输出
```

```
    MOV   AL, 0DH
    OUT   DX, AL                        ;使 PC6 置 1(INTE=1),允许 8255 产生中断
    MOV   AL, 00H
    MOV   DX, 0FBC0H
    OUT   DX, AL
    MOV   AX, OFFSET DATA               ;从 A 口输出一个空字符,引发第一次中断
    MOV   STR_PTR, AX                   ;设置字符串偏移地址
    MOV   AX, SEG DATA
    MOV   STR_PTR+2, AX                 ;设置字符串段地址
    STI                                 ;开中断
    ⋮
```

中断服务子程序如下：

```
PRINT: PUSH  SI
       PUSH  AX
       PUSH  DS
       LDS   SI, DWORD PTR STR_PTR
NEXT:  LODSB                        ;取一个字符
       MOV   STR_PTR,SI             ;保存新的串指针
       MOV   DX, 0FBC0H
       OUT   DX, AL                 ;输出字符到 8255 的 A 口
       MOV   AL, 20H
       MOV   DX, 0FF00H             ;8259A 的 OCW2
       OUT   DX, AL                 ;送中断结束命令给 8259A
       POP   DS
       POP   AX
       POP   SI
       IRET                         ;中断返回
```

【例 7-5】 用 8255 并行接口芯片实现键盘接口，其电路如图 7-34 所示。图中，按键排列成 4 行 4 列，8255 的 C 口设置为方式 0，并将 $PC_7 \sim PC_4$ 设定为输出，与各行线相连；$PC_3 \sim PC_0$ 设定为输入，与各列线相连。

题目分析：

键盘输入是微机系统最常用的输入方式。键盘的结构有两种形式：线性键盘和矩阵键盘。线性键盘就是若干独立的开关(按键)，每个按键将其一端直接与微机某输入端口的一位相连，另一端接地，就可完成硬件的连接。其接口程序也很简单，只要查询该输入端口各位的状态，即可判别是否有键按下以及按下的是哪一个键。线性键盘有多少按键，就有多少根连线与微机输入端口相连，因此只适用于按键少的应用场合。

矩阵键盘的按键排成 n 行 m 列的矩阵形式，每个按键占据行列的一个交点，需要的连接线数是 $n+m$ 根，容许的最大按键数是 $n\times m$ 个。矩阵键盘所需的连线数非常少，是一般微机常用的键盘结构。矩阵键盘按键的识别主要有扫描法和反转法两种。下面以

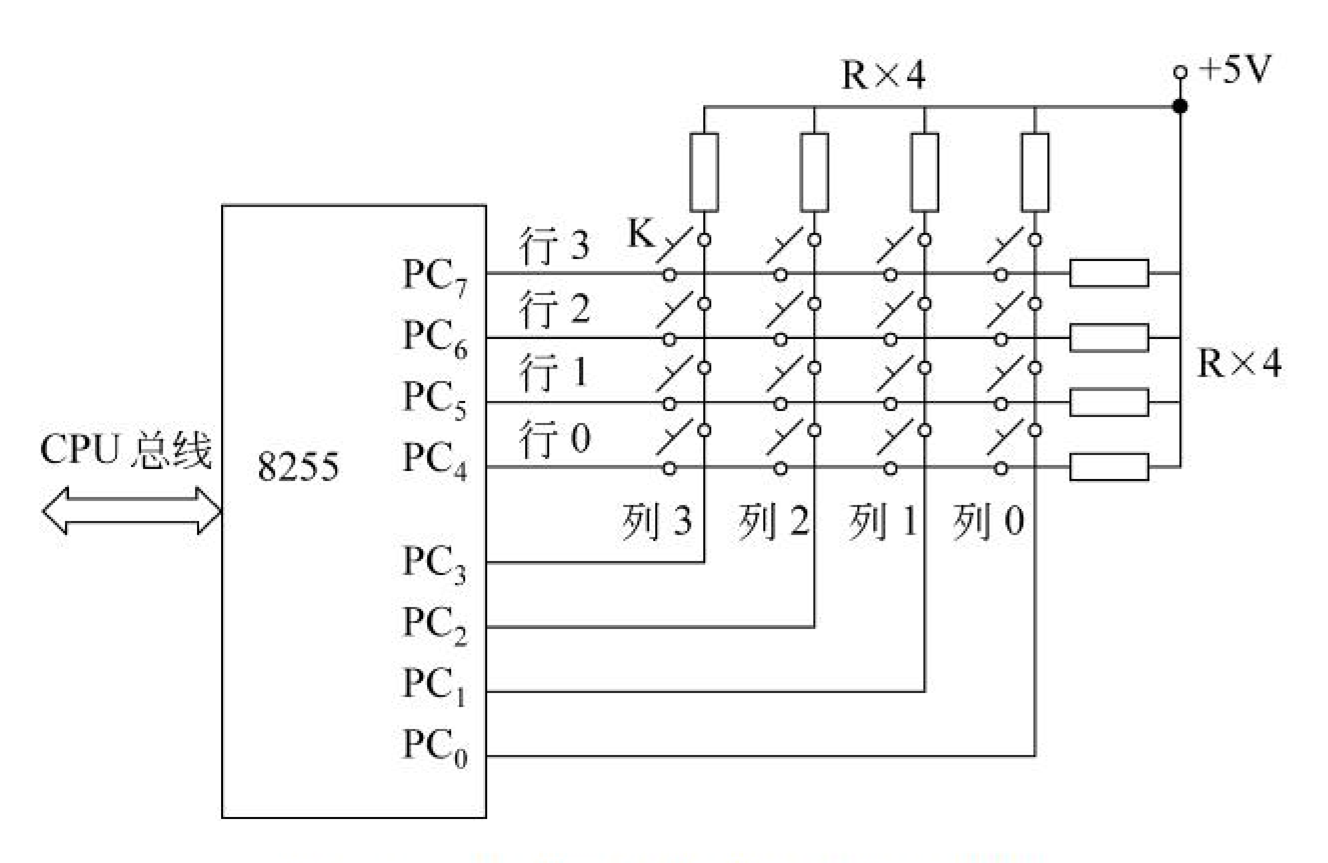


图 7-34　矩阵式键盘接口电路原理图

4×4 矩阵键盘为例来说明用 8255 作为矩阵键盘接口的原理及按键识别方法。

(1) 扫描法。扫描法就是逐行输出 0,然后读入列值,并检查有无为 0 的位(与某一列相对应)。若有,则当前行该列的键被按下。实际应用中往往采用一些技巧来加快扫描速度。用以下 3 个步骤即可检查出哪一个键被按下。

① 识别有键按下否。PC_7～PC_4 输出全 0,然后从 PC_3～PC_0 读入,若读入的数据中有一位为 0,则表明有某个键被按下,转第②步,否则在本步骤中循环。

② 去抖动。延时 20ms 左右,过滤掉按键的抖动,然后按第①步的方法再做一次,若还有键闭合,则认为确实有一个键被按下,否则返回第①步。

③ 查找被按下的键。从第 0 行开始,顺序逐行扫描,即逐行输出 0。每扫描一行,读入列线数据,若数据中有一位为 0,则表示该位对应的列与当前扫描行的交点处的按键被按下。

(2) 反转法。此法不需要逐行扫描,仅用两步即可找到按下的键,步骤如下。

① 将 PC_7～PC_4 设定为输出,PC_3～PC_0 设定为输入。然后向行线输出全 0(即 PC_7～PC_4 输出全 0),接着从 PC_3～PC_0 读入列线的值,若读入的数据中有一位为 0,则表明与该位对应的列线上有某个键被按下,存储此值作为“列值”,转第②步,否则在本步骤中循环。

② 将 PC_7～PC_4 设定为输入,PC_3～PC_0 设定为输出。把第①步读入的值再输出到列线上(即把“列值”从 PC_3～PC_0 输出),接着从 PC_7～PC_4 读入行线的值,其中必有一位为 0,为 0 的位所对应的行线就是被按键所在的行,存储此值作为“行值”。将行值和列值组合在一起,用查表的方法即可得到按键的键号。

例如,若第 0 行第 2 列(0,2)的键按下,则第①步从列线读回的列值为 1011B;第②步中再将 1011B 从列线输出,从行线读回的行值为 1110B,二者组合,得到该键的行列值组合为 11101011B。

因为在键盘扫描过程中要反转行线与列线的输入输出方向,所以此法被称为反转法。

以下是与图 7-34 相对应的采用反转法的按键识别程序。设 8255 端口 A 的地址为

40H，端口B的地址为41H，端口C的地址为42H，控制寄存器地址为43H。

```
START: MOV   AL, 10000001B         ;方式0,C口高4位输出,低4位输入
       OUT   43H, AL
       MOV   AL, 0
       OUT   42H, AL               ;各行线(PC7~PC4)为0
WAIT1: IN    AL, 42H               ;读入列线(PC3~PC0)状态
       AND   AL, 0FH               ;保留低4位
       CMP   AL, 0FH               ;检查有键按下否(是否存在为0的位)
       JE    WAIT1                 ;全1表示无按键,循环继续检测
       MOV   AH, AL                ;保存列值
       MOV   AL, 10001000B         ;方式0,C口高4位输入,低4位输出
       OUT   43H, AL               ;反转输入输出方向
       MOV   AL, AH
       OUT   42H, AL               ;把列值反向输出到列线上
       IN    AL, 42H               ;读入行线(PC7~PC4)状态
       AND   AL, 0F0H              ;保留高4位
       OR    AL, AH                ;组合行值和列值

       <查表求出按键的键号>
          …
```

用扫描法获取按键值的程序作为练习由读者自行编写。

【例7-6】 PC/XT微机中8255的连接。

在PC/XT中，系统板上的外围接口电路主要是由可编程接口芯片8255A-5以及相关电路组成的，其连接示意图如图7-35所示。

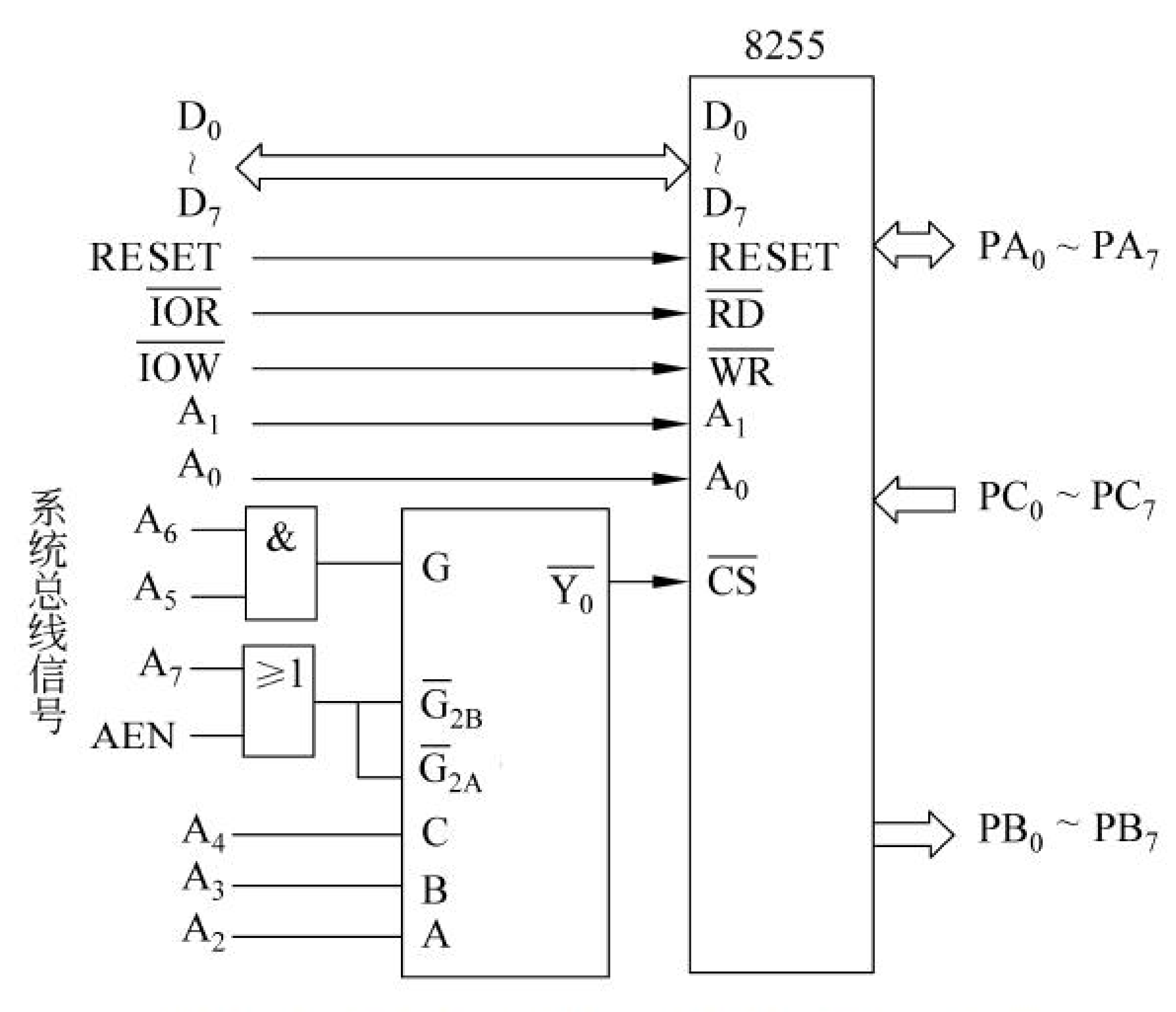


图7-35 PC/XT中的8255A-5连接示意图

PC/XT微机中，8255A的端口地址范围为60H～63H。A、B、C这3个口均工作于

方式 0。A 口在加电自检时工作在输出状态，输出当前被检测部件的标识信号。此时的 B 口也工作于输出状态，而 C 口工作为输入状态，因此其方式控制字为 89H。

在正常工作时，A 口作为输入口，用来读取键盘扫描码；B 口和 C 口仍分别为输出和输入口。B 口用于输出系统内部的控制信号，控制系统板部分电路的动作，如定时器、扬声器、键盘，允许 RAM 奇偶校验、允许 I/O 通道校验以及控制系统配置开关信号的读取；C 口用来读取系统内部的状态信号，包括系统配置开关的状态、8253 的 OUT_2、I/O 通道奇偶校验和 RAM 奇偶校验的状态等。此时的控制字为 99H。

学习完了可编程并行接口 8255，可以考虑修改我们的"家庭安全防盗系统"设计了。毕竟相对于简单接口，可编程接口有着更广泛、更便利的应用。

"家庭安全防盗系统"设计方案示例 3：

设计同样基于这样的假设：监测装置输出电平信号。当出现异常时，监测装置输出高电平(1)，正常状态则输出低电平(0)。

基本方案：与"设计方案示例 2"一样，可以利用定时/计数器 8253 控制报警器发声。但是考虑到简单接口芯片功能较弱，可以选择利用可编程并行接口芯片 8255 的 PA 端口来获取监测装置的输出，在 PC 口的高 4 位中选择一位控制报警灯闪烁，另一位作为 8253 芯片的启动控制信号，如图 7-36 所示。

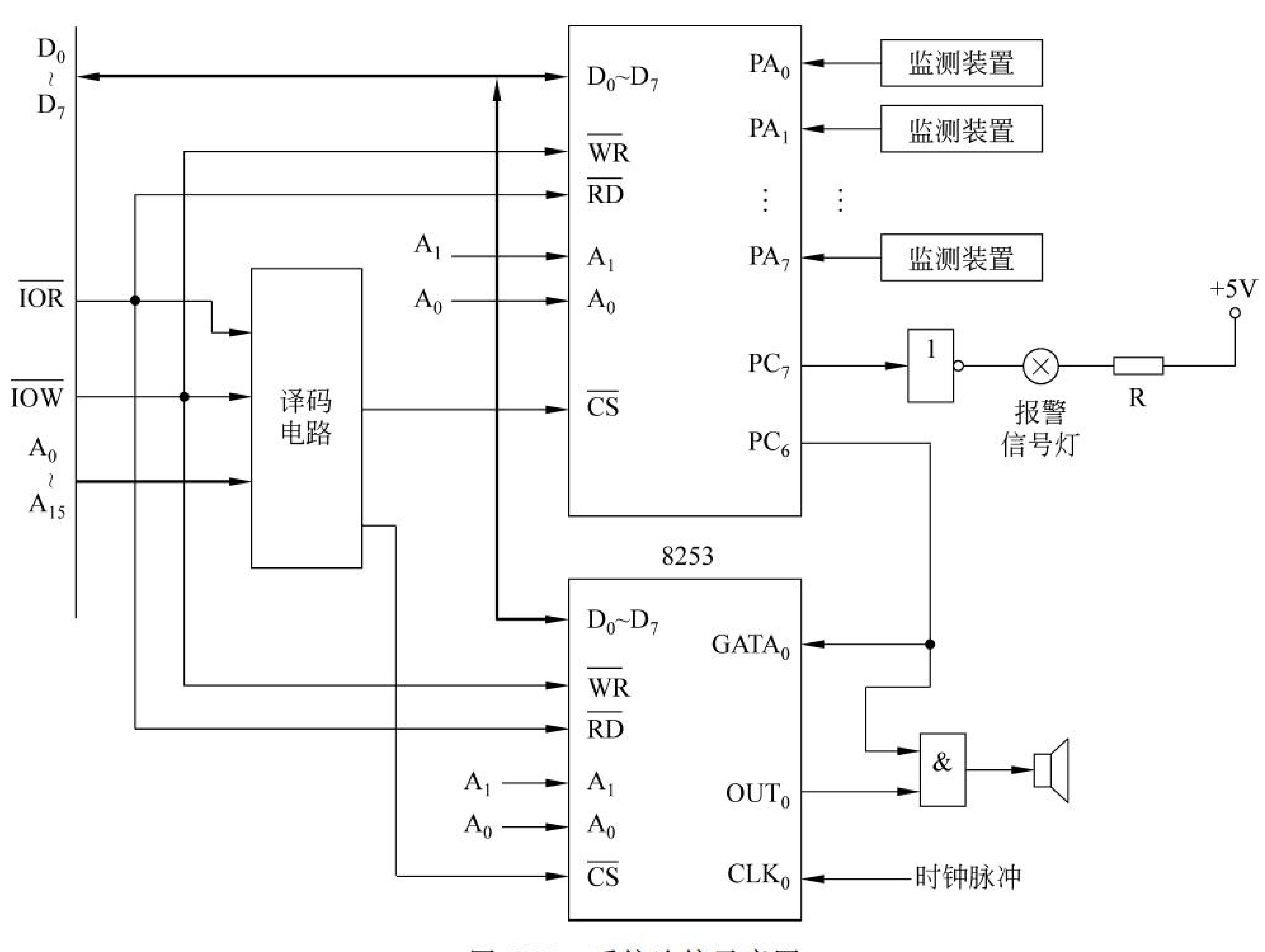


图 7-36　系统连接示意图

这样的设计利用了两片可编程接口芯片。这个方案如何完善？你可以尝试一下！

7.4　可编程串行接口 8250

Intel 8250 是专用于异步串行通信的可编程串行接口芯片，具有很强的串行通信能力和灵活的可编程性能，在微机中的应用极为广泛。

7.4.1　8250 的外部引线及功能

可编程串行通信接口 8250 的外部引线图如图 7-37 所示，共有 40 根引脚，单电源 +5V 供电。除电源线(V_{CC})和地线(GND)外，其引脚信号可分为面向系统和面向外部通信设备两大类。

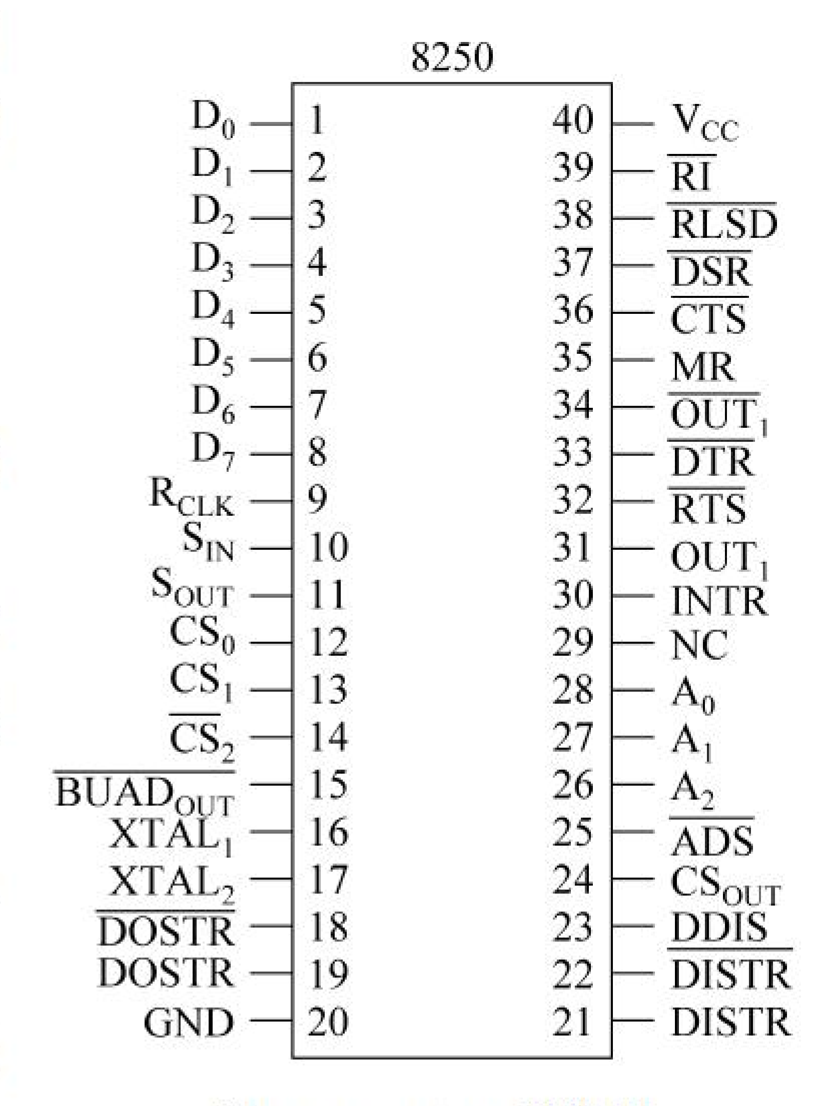


图 7-37　8250 引线图

1. 面向系统的引脚信号

(1) $D_0 \sim D_7$：双向数据线。$D_0 \sim D_7$ 与系统数据总线相连接，用以传送数据、控制信息和状态信息。

(2) CS_0、CS_1、$\overline{CS_2}$：片选信号，输入。只有当它们同时有效，即 $CS_0=1$，$CS_1=1$，$\overline{CS_2}=0$ 时，才能选中该 8250 芯片。

(3) CS_{OUT}：片选输出信号。当 8250 的 CS_0、CS_1 和$\overline{CS_2}$同时有效时，CS_{OUT}为高电平。

(4) $A_0 \sim A_2$：8250 内部寄存器的选择信号。它们的不同编码，可以选中 8250 内部不同的寄存器。详细情况在下面再做介绍。

(5) $\overline{ADS}$：地址选通信号，低电平有效。$\overline{ADS}$有效时可将 CS_0、CS_1、$\overline{CS_2}$及 A_0、A_1、A_2 锁存于 8250 内部。若在工作中不需要锁存上述信号，则可将 ADS 直接接地，使其恒有效。

(6) DISTR、$\overline{DISTR}$：数据输入选通信号。当它们其中任何一个有效时(DISTR 为高或$\overline{DISTR}$为低)，被选中的 8250 寄存器内容可被读出。它们经常与系统总线上的$\overline{IOR}$信号相连接。当它们同时无效时，8250 不能读出。

(7) DOSTR、$\overline{DOSTR}$：数据输出选通信号。当它们其中一个有效时(DOSTR 为高电平或$\overline{DOSTR}$为低电平)，被选中的 8250 寄存器可写入数据或控制字。它们常与系统总线的$\overline{IOW}$相连。当它们同时无效时，8250 则不能写入。

(8) DDIS：驱动器禁止信号。该输出信号在 CPU 读 8250 时为低电平，非读时为高电平。可用此信号来控制 8250 与系统总线间的数据总线驱动器。

(9) INTR：中断请求输出信号，高电平有效。当 8250 中断允许时，接收出错、接收数据寄存器满、发送数据寄存器空以及 MODEM 的状态均能够产生有效的 INTR 信号。主复位信号(MR)可使该输出信号无效。

（10）MR：主复位输入信号，高电平有效。MR 通常与系统复位信号 RESET 相连。主复位时，除了接收数据寄存器、发送数据寄存器和除数锁存器外，其他内部寄存器及信号均受到主复位的影响，详细情况如表 7-6 所示。

表 7-6　MR 功能

寄存器或信号	复 位 控 制	复位后的状态
通信控制寄存器	MR	各位均为低电平
中断允许寄存器	MR	各位均为低电平
中断标识寄存器	MR	第 0 位高电平，其余均为低
MODEM 控制寄存器	MR	各位均为低电平
通信状态寄存器	MR	除第 5、6 位外其余均为高
INTR（线路状态错）	读通信状态寄存器或 MR	低电平
INTR（发送寄存器空）	读中断标志寄存器，写发送数据寄存器或 MR	低电平
INTR（接收寄存器满）	读接收数据寄存器或 MR	低电平
INTR（MODEM 状态改变）	读 MODEM 状态寄存器或 MR	低电平
S_{OUT}	MR	高电平
OUT_1、OUT_2、RTS、DTR	MR	高电平

2．面向外部通信设备的引脚信号

（1）S_{IN}：串行数据输入端。外设或其他系统传送来的串行数据由该端进入 8250。

（2）S_{OUT}：串行数据输出端。主复位信号可使其变为高电平。

（3）$\overline{CTS}$：清除发送信号。输入，低电平有效。当它有效时表示提供 CTS 信号的设备可以接收 8250 发送的数据，它是提供 CTS 信号的设备向 8250 发出的 RTS 信号的应答信号。

（4）$\overline{RTS}$：请求发送信号。输出，低电平有效。它是 8250 向外设发出的发送数据请求信号。它与$\overline{DTR}$信号具有同样的功能。

（5）$\overline{DTR}$：数据终端准备好信号。输出，低电平有效。它表示 8250 已准备好，可以接收数据。

（6）$\overline{DSR}$：数据装置准备好信号。输入，低电平有效。它表示接收数据的外设已准备好接收数据。它是对$\overline{DTR}$信号的应答。

（7）$\overline{RLSD}$：接收线路信号检测信号。输入，低电平有效，$\overline{RLSD}$表示 MODEM 已检测到数据载波信号。

（8）$\overline{RI}$：振铃指示信号。输入，低电平有效。$\overline{RI}$表示 MODEM 已接收到一个电话振铃信号。

（9）$\overline{OUT_1}$：可由用户编程确定其状态的输出端。若用户在 MODEM 控制寄存器第二位（OUT_1）写入 1，则$\overline{OUT_1}$输出端变为低电平。主复位信号（MR）可将$\overline{OUT_1}$置为高电平。

（10）$\overline{OUT_2}$与$\overline{OUT_1}$一样，也可由用户编程指定。只是要将 MODEM 控制寄存器的第三位（OUT_2）写入 1，就可使$\overline{OUT_2}$变为低电平。主复位信号（MR）可将其置为高电平。

（11）$\overline{BAUD_{OUT}}$：波特率信号输出。该端输出的是主参考时钟频率除以 8250 内部除

数寄存器中的除数后所得到的频率信号。这个频率信号就是 8250 的发送时钟信号，是发送数据波特率的 16 倍。若将此信号接到 R_{CLK} 上，又可以同时作为接收时钟使用。

（12）$XTAL_1$、$XTAL_2$：外部时钟端。这两端可接晶振或直接接外部时钟信号。

（13）R_{CLK}：接收时钟信号。该输入信号的频率为接收数据波特率的 16 倍。

7.4.2 8250 的结构及内部寄存器

8250 的内部结构框图如图 7-38 所示。

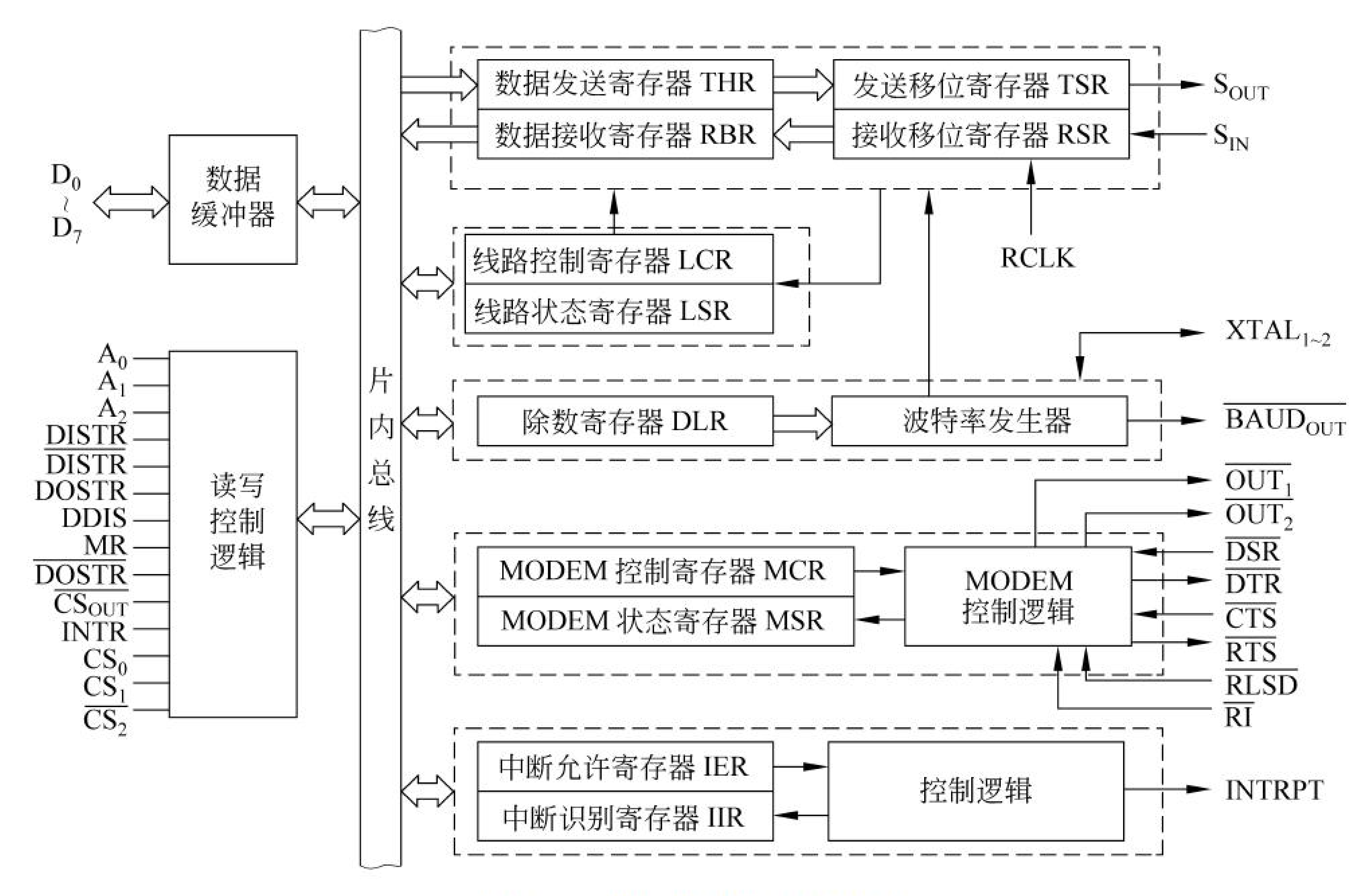


图 7-38 8250 的内部结构框图

由图可知，8250 中除与系统相连的数据缓冲器、读写控制逻辑外，还包括 10 个寄存器(可分为 5 个功能模块)。程序员在对 8250 编程时要经常与这些寄存器打交道，所以要使用 8250 就必须熟练掌握它们各位的意义和使用方法。下面分别介绍 8250 的这 10 个内部寄存器各位的功能。

1. 数据发送寄存器 THR

数据发送寄存器 THR 是一个 8 位的寄存器。发送数据时，CPU 将数据写入 THR。只要 TSR 空，THR 中的数据便会由 8250 的硬件自动送入 TSR 中，以便串行移出。

2. 数据接收缓冲寄存器 RBR

数据接收缓冲寄存器 RBR 是一个 8 位的寄存器。8250 接收到一个完整的字符时，便会把该字符从接收移位寄存器 RSR 传送到 RBR。CPU 可从由 RBR 读出接收到的数据。

3. 通信线路控制寄存器 LCR

通信线路控制寄存器 LCR 是一个 8 位的寄存器，其各位的主要功能如图 7-39 所示。

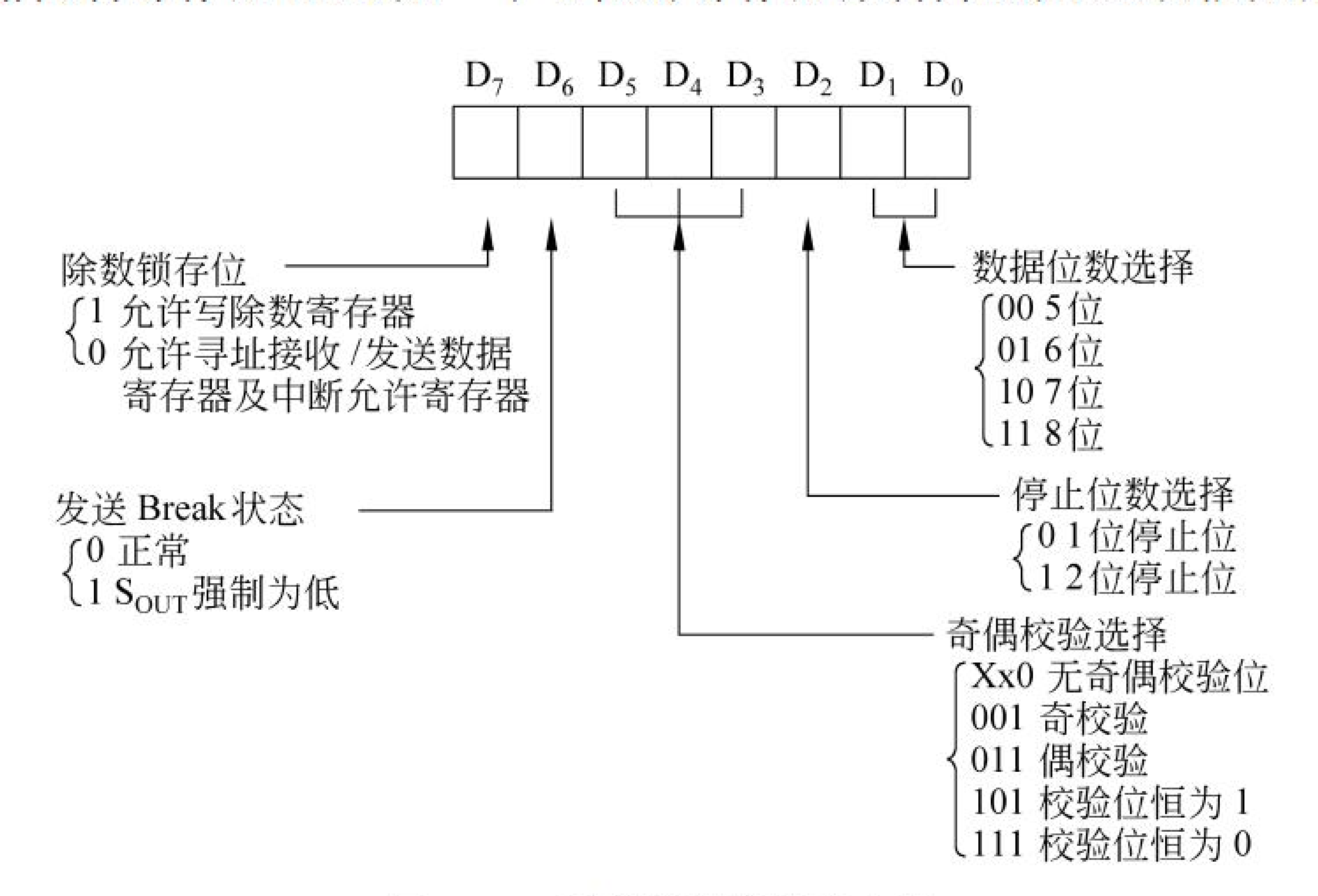


图 7-39　**通信线路控制寄存器**

LCR 主要用于决定在串行通信时所使用的数据格式，例如数据位数、奇偶校验及停止位的多少等。因芯片仅有 3 根地址线，最多只能寻址 8 个寄存器，为此只好使两个除数寄存器和其他寄存器共用地址。当前是寻址除数寄存器还是其他寄存器，是由 LCR 的最高位 D_7 来区分的。当需要读写除数寄存器时，必须先使 LCR 的 D_7 置 1，而在读写其他寄存器时，又必须先将其设为 0。

4. 通信线路状态寄存器 LSR

LSR 是一个 8 位寄存器，其各位的功能如图 7-40 所示。它存放了通信过程中 8250 接收和发送数据的状态。

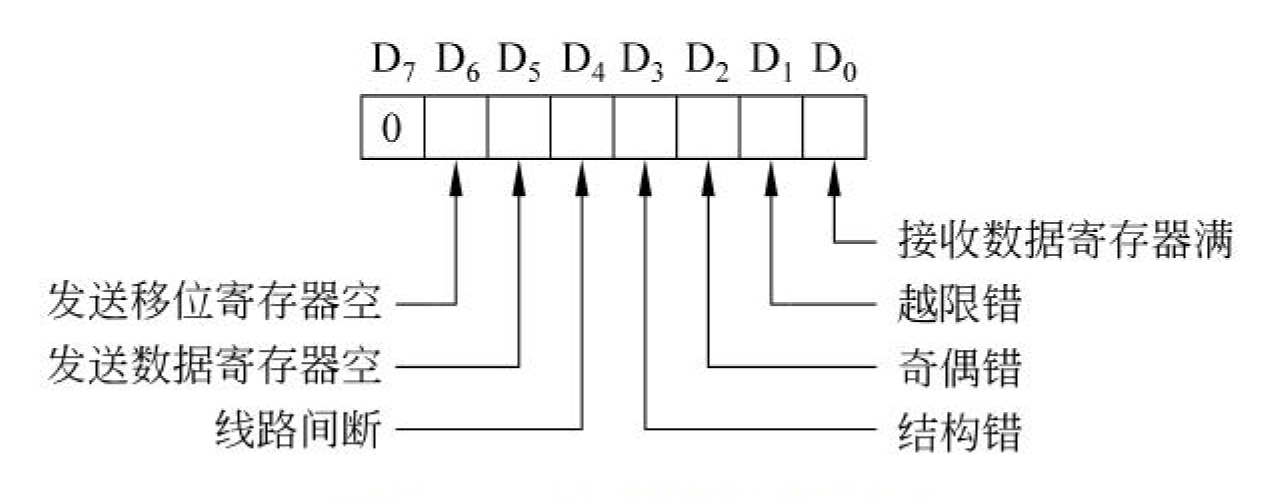


图 7-40　**通信线路状态寄存器**

(1) D_0：此位为 1 时，表示 8250 已接收到一个完整的字符，CPU 可以从 8250 的接收数据寄存器中读取。一旦读取后，此位即变为 0。

(2) D_1：越限状态错标志。接收数据寄存器中的前一数据还未被 CPU 读走，而后一个数据已经到来并将其破坏时，此位为 1。当 CPU 读接收数据寄存器时使此位变为 0。

(3) D_2：奇偶校验错标志。在 8250 对收到的一个完整的字符编码进行奇偶校验时，

若发现其值与规定的奇偶校验不同，则使此位为 1，表示数据可能有错。当 CPU 读 LCR 时此位变为 0。

(4) D_3：结构错标志。当接收到的数据停止位个数不正确时，此位置 1。当 CPU 读 LCR 时此位变为 0。

(5) D_4：线路间断标志。若在一个完整的字符编码的时间间隔中收到的均为空闲状态，则此位置 1，表示线路信号间断。当 CPU 读通信状态寄存器时使此位变为 0。

出现以上 4 种状态中的任何一种都会使 8250 发出线路状态错中断。

(6) D_5：此位为 1 表示数据发送保持寄存器 THR 空。CPU 将数据写入 THR 后，此位清 0。

(7) D_6：此位为 1 表示发送移位寄存器 TSR 空。当 THR 的数据送入 TSR 时，此位清 0。

(8) D_7：此位恒为 0。

5. 除数寄存器 DLR

DLR 是一个 16 位的寄存器。外部时钟按 DLR 中的除数(分频系数)进行分频，可以获得所需的波特率。如果外部时钟频率 f 已知，而 8250 所要求的波特率 B 也已规定，那么就可以由下式求出 DLR 中除数的值：

$$除数 = f/(B \times 16)$$

通常，8250 使用 1.8432MHz 的基准时钟输入，所以上式可写为

$$除数 = 1\ 843\ 200/(B \times 16)$$

例如，若要求使用 1200b/s 来传送数据，则可计算出除数应为 96。在初始化 8250 时，最开始就应将除数写到 DLR 中，以便产生所希望的波特率。为了写入除数，应首先把 LCR 的 D_7 置 1，然后将 16 位除数按先低 8 位、后高 8 位的顺序写入 DLR。写完后，还应把 LCR 的 D_7 再置为 0，以便 8250 进行正常操作。

6. modem 控制寄存器 MCR

MCR 是一个 8 位的寄存器，用来对 modem 实施控制。其中高 3 位恒为 0，其余各位的功能如图 7-41 所示。

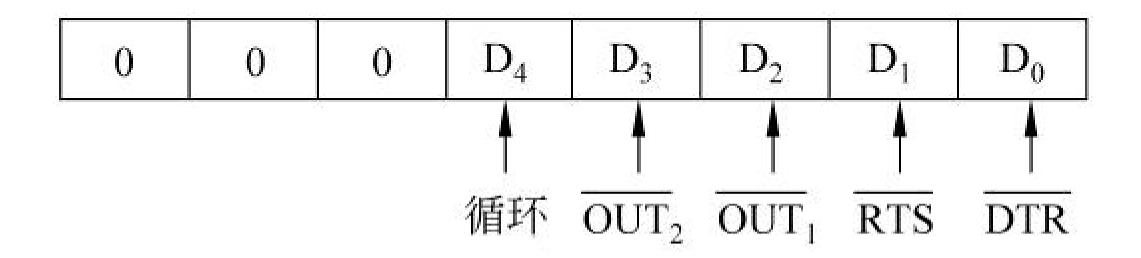


图 7-41 modem 控制字

- D_0：此位用于设置数据终端准备好信号。当它为 1 时，使 8250 的 $\overline{DTR}$ 输出为低，表示 8250 准备好接收数据；当它为 0 时，使 8250 的 $\overline{DTR}$ 输出为高，表示 8250 没有准备好。
- D_1：此位为 1 时，8250 的 $\overline{RTS}$ 输出低电平，表示 8250 已准备好发送数据；当它为 0 时，$\overline{RTS}$ 输出高电平，表明 8250 未准备好发送。

• D_2、D_3：这两位分别用以控制 8250 的输出线 OUT_1 和 OUT_2。当它们为 1 时，对应的 OUT 输出为 0；当它们为 0 时，对应的 OUT 输出为 1。

• D_4：用于环回检测控制，实现 8250 的自我环回测试。当 $D_4=1$ 时，S_{OUT} 为高电平状态，而 S_{IN} 将与系统相分离。这时 TSR 的数据将由 8250 内部直接回送到 RSR 的输入端。modem 用以控制 8250 的 4 个信号 CTS、DSR、RLSD 和 RI 与系统分离。同时，8250 用来控制 modem 的 4 个输出信号 RTS、DTR、OUT_1 和 OUT_2 在 8250 芯片内部与 $\overline{CTS}$、$\overline{DSR}$、$\overline{RLSD}$ 和 $\overline{RI}$ 相连接，实现数据在 8250 芯片内部的自发自收。这样，8250 发送的串行数据在其内部被接收，从而完成 8250 的自检，并且在完成自测试过程中不需要外部连线。在自回环测试时，中断仍能产生。值得注意的是，在这种情况下，modem 状态中断是由 modem 控制寄存器提供的。

当 $D_4=0$ 时，8250 正常工作。从环回测试转到正常工作状态，必须对 8250 重新初始化，其中包括将 D_4 清零。

7. modem 状态寄存器 MSR

MSR 用来反映 8250 与通信设备之间应答联络输入信号的当前状态以及这些信号的变化情况。其状态字的格式如图 7-42 所示。

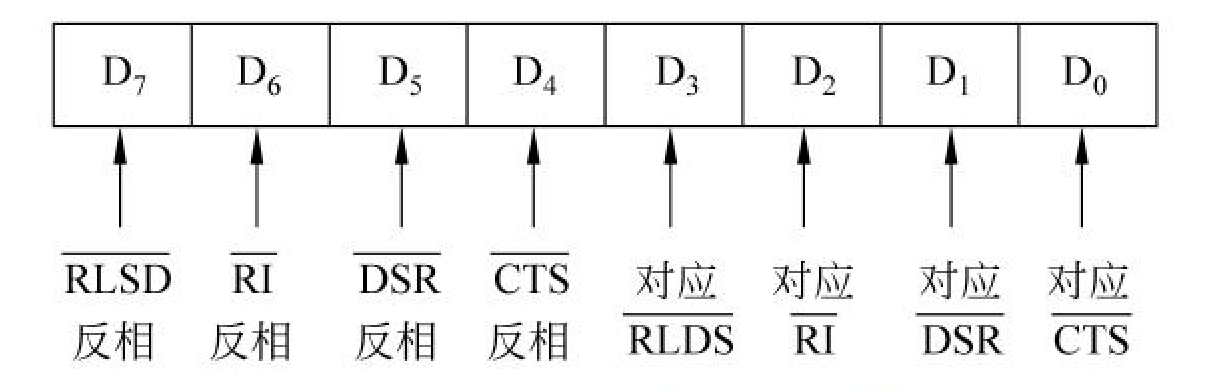


图 7-42 modem 状态寄存器

MSR 的低 4 位是应答输入信号发生变化(从高变低或从低变高)的状态标志，CPU 读 MSR 时，把这 4 位同时清零。这 4 位分别对应 $\overline{CTS}$、$\overline{DSR}$、$\overline{RI}$ 和 $\overline{RLSD}$。当某位为 1 时，表示从上次读 MSR 后，相应的应答输入信号发生了变化。当某位为 0 时，则说明相应的应答输入信号状态无改变。

MSR 的高 4 位反映了 $\overline{CTS}$、$\overline{DSR}$、$\overline{RI}$ 和 $\overline{RLSD}$ 这 4 个输入信号的当前状态。

(1) D_4 是 $\overline{CTS}$ 反相之后的状态，自测试时为 RTS 的状态。

(2) D_5 是 $\overline{DSR}$ 反相之后的状态，自测试时为 DTR 的状态。

(3) D_6 是 $\overline{RI}$ 反相之后的状态，自测试时为 OUT_1 的状态。

(4) D_7 是 $\overline{RLSD}$ 反相之后的状态，自测试时为 OUT_2 的状态。

8. 中断允许寄存器 IER

IER 只使用 $D_0 \sim D_3$ 这 4 位，高 4 位不用。$D_0 \sim D_3$ 每位的 1 或 0 分别用于允许或禁止 8250 的 4 个中断源发出中断请求，其格式如图 7-43 所示。

如果 IER 的 $D_0 \sim D_3$ 均为 0，则禁止 8250 发出中断。在 IER 中，接收线路状态引起

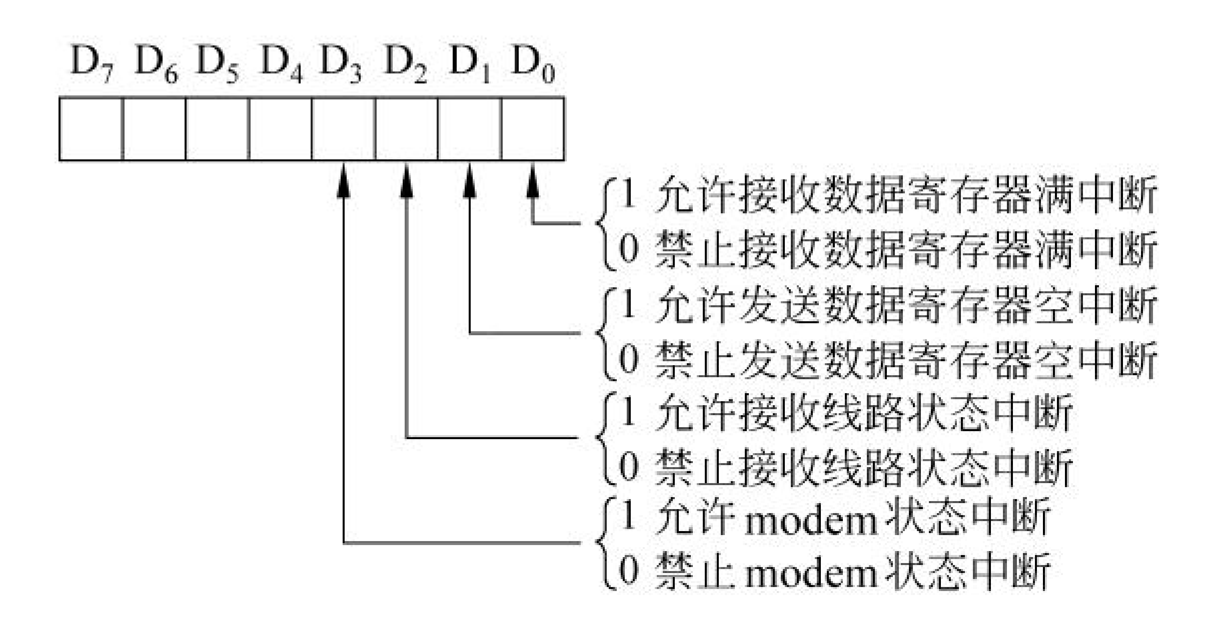


图 7-43　中断允许寄存器

的中断包括越限错、奇偶错、结构错和间断。对于 modem 状态引起的中断见下面对 modem 状态寄存器的解释。

9. 中断识别寄存器 IIR

IIR 是一个 8 位的寄存器，其高 5 位恒为 0，只使用低 3 位作 8250 的中断识别标志，格式如图 7-44 所示。8250 有 4 个中断源，它们的中断优先级顺序如下。

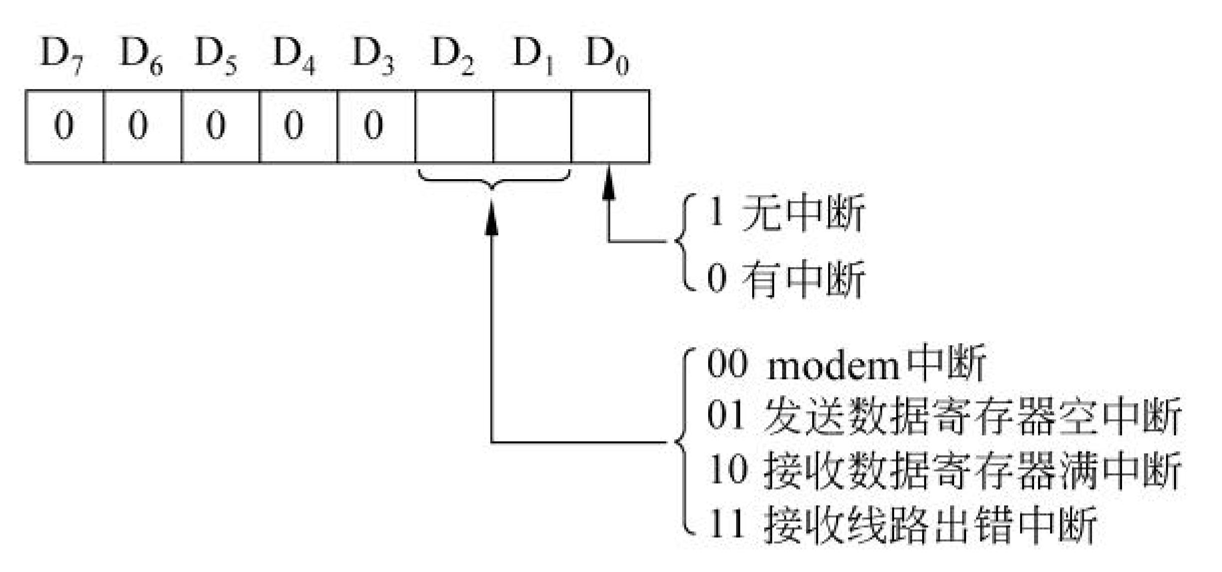


图 7-44　中断识别寄存器

(1) 接收器线路状态中断为最高优先级，包括越限错、奇偶错、结构错和间断。读 LSR 可使此中断复位。

(2) 第二是接收数据缓冲寄存器满中断。读 RBR 可复位此中断。

(3) 第三为发送数据保持寄存器空中断。写 THR 可复位此中断。

(4) 最低优先级为 modem 状态中断，包括 CTS、DSR、RI、DCD 等 modem 状态中断源。读 modem 状态寄存器可复位此中断。

7.4.3　8250 的工作过程

1. 数据发送过程

CPU 将要发送的数据以字符为单位写到 8250 的 THR 中，如图 7-38 所示。当 TSR 中的数据全部移出变空时，存于 THR 中待发送的数据就会自动并行送到 TSR①。

① 8250 初始化后，TSR 为空状态，所以初始化后传送到 THR 的第一个字符总是立即送入 TSR。

TSR 在发送时钟的激励下，按照事先和接收方约定的字符传送格式，如图 7-3 所示，加上起始位、奇偶校验位和停止位，再以约定的波特率(由波特率控制部分产生)按照从低到高的顺序一位接一位地由 S_{OUT} 端发送出去。

一旦 THR 的内容送到 TSR，就会在 LSR 中建立“数据发送保持寄存器空”的状态位；而且也可以用此状态位来触发产生中断。因此，查询该状态位或者利用该状态触发的中断即可实现数据的连续发送。

2. 数据接收

由通信对方来的数据在接收时钟 R_{CLK} 作用下，通过 S_{IN} 端逐位进入 RSR；RSR 根据初始化时定义的数据位数确定接收到了一个完整的数据后会立即将数据自动并行传送到 RBR；RBR 收到 RSR 的数据后，就立即在状态寄存器中建立“接收数据准备好”的状态，而且也可以用此状态位来触发中断。因此，查询该状态位或者利用该状态触发的中断即可实现数据的连续接收。

由于串行异步通信的速率较低，无论是用查询方式或中断方式来实现异步通信均不很困难。

7.4.4 8250 的应用

1. 8250 的寻址和连接

一片 8250 芯片共占用 7 个端口地址。表 7-7 详细列出了各内部寄存器具体的地址安排，另外还列出了 IBM PC/XT 中异步串行通信口 COM1 各寄存器的物理地址(COM2 的物理地址相应为 2F8H～2FFH)。

表 7-7　8250 内部寄存器寻址

CS_0	CS_1	$\overline{CS_2}$	A_2	A_1	A_0	DLAB	COM1 地址	寄　存　器
1	1	0	0	0	0	0	3F8H	发送保持寄存器 THR(写)，接收缓冲寄存器 RBR(读)
1	1	0	0	0	0	1	3F8H	除数锁存器(低 8 位)DLL
1	1	0	0	0	1	1	3F9H	除数锁存器(高 8 位)DLH
1	1	0	0	0	1	0	3F9H	中断允许寄存器 IER
1	1	0	0	1	0	×	3FAH	中断识别寄存器 IIR
1	1	0	0	1	1	×	3FBH	通信线路控制寄存器 LCR
1	1	0	1	0	0	×	3FCH	MODEM 控制寄存器 MCR
1	1	0	1	0	1	×	3FDH	通信线路状态寄存器 LSR
1	1	0	1	1	0	×	3FEH	MODEM 状态寄存器 MSR
1	1	0	1	1	1	×	3FFH	(无效)

8250 内部有 10 个与编程使用有关的寄存器，可利用片选信号 CS_0、CS_1 和 $\overline{CS_2}$ 选中 8250，利用芯片上 A_0、A_1、A_2 这 3 条地址线的 8 种不同编码选择 8 个寄存器，再利用通信控制字的最高位——除数锁定位(DLAB)来选中除数锁存器。由于有的寄存器是只写

的，有的寄存器是只读的，故还可以利用读写信号来加以选择。通过上述这些办法，就可以对指定的寄存器进行寻址访问。

在 PC 中，串行通信接口由 8250 来实现，图 7-45 表示了它与总线的连接。

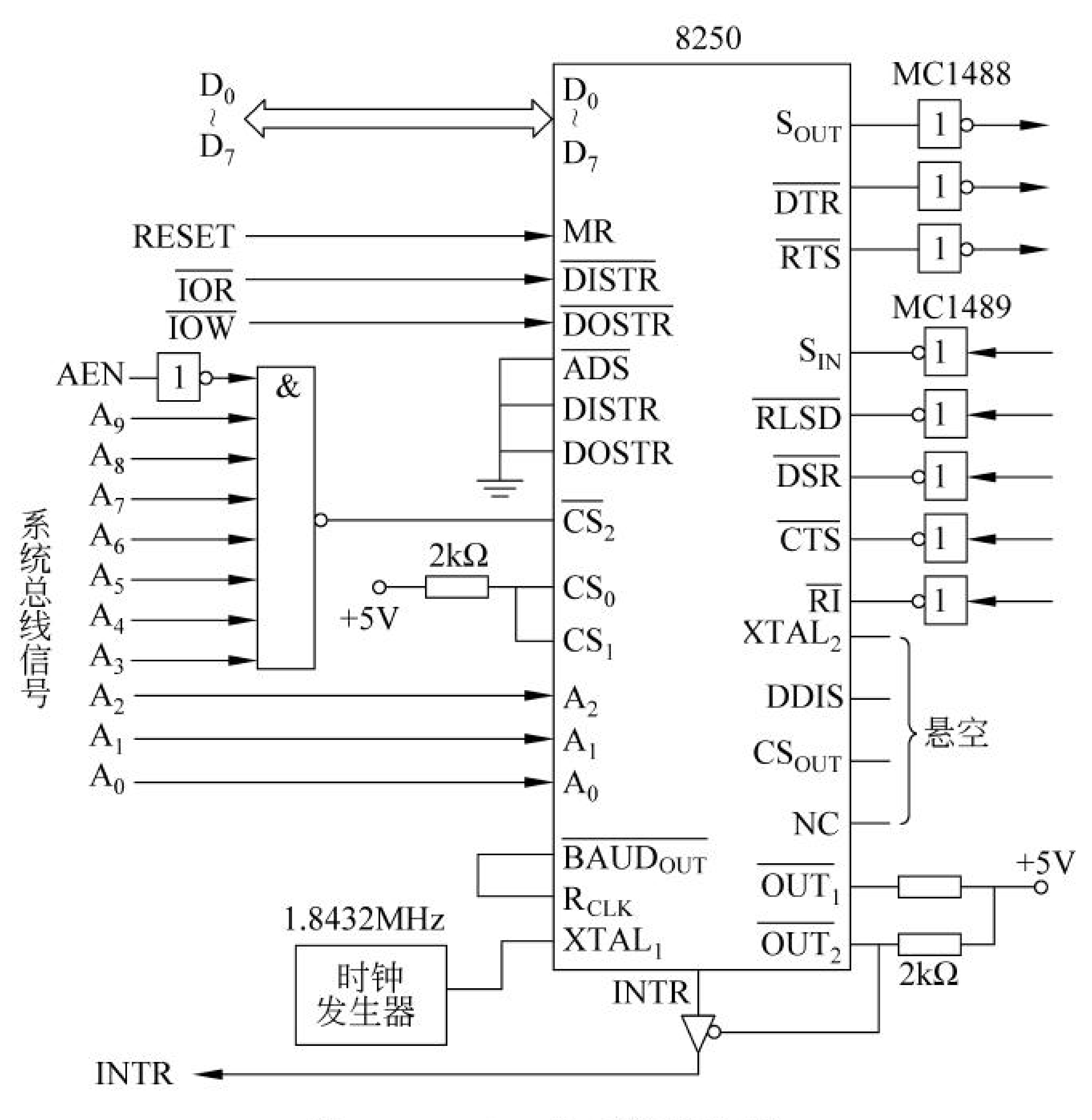


图 7-45　8250 与系统的连接

由图 7-45 可知，8250 的地址由 10 条地址线来决定，其地址范围为 3F8H～3FFH(COM1)。在寻址 8250 时，AEN 信号总处于低电平。由于 $\overline{ADS}$ 始终接地，CS_0 和 CS_1 接高电平，故只要地址译码输出使 $\overline{CS_2}$ 为低电平即可选中 8250。再利用表 7-7 所示的寻址方法，就可对 8250 的 9 个内部寄存器寻址。

时钟发生器将外部时钟信号由 $XTAL_1$ 加到 8250 上，而其 $\overline{BAUD_{OUT}}$ 输出又作为接收时钟加到 R_{CLK} 上。芯片上的一些引线固定接高电平或接地，而一些不用的则悬空。这是 8250 在电路连接上为我们提供的灵活性。

2. 8250 的初始化及应用

8250 初始化时，通常首先使通信控制字的 $D_7=1$，即使 DLAB 为 1。在此条件下，将除数低 8 位和高 8 位分别写入 8250 内部的除数寄存器。然后再以不同的地址分别写入通信控制字、modem 控制字及中断允许字等。其具体做法可按图 7-46 所示的流程依次进行。现以图 7-45 为例，对 8250 进行初始化编程。

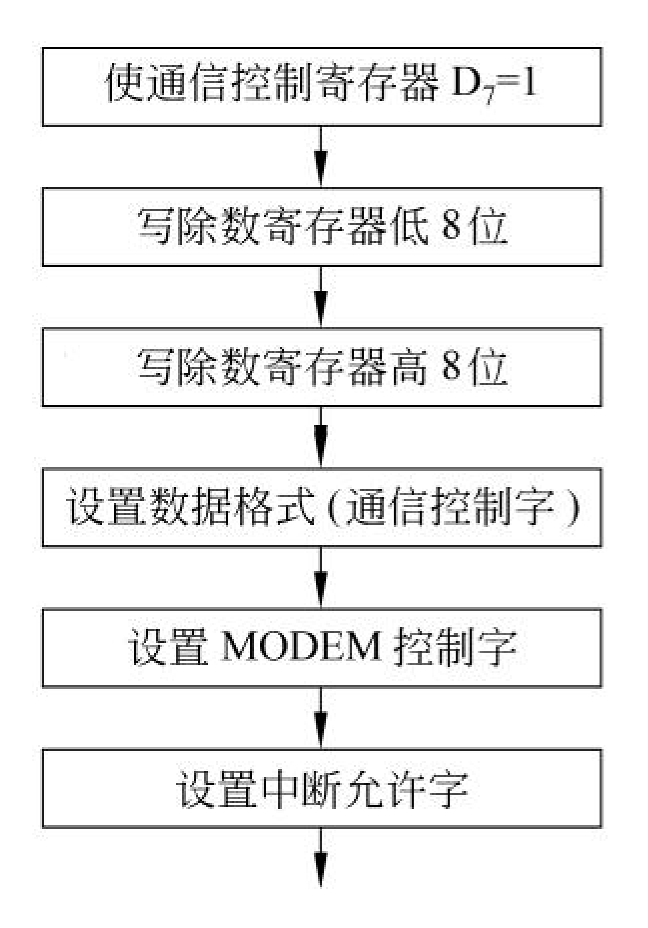


图 7-46　8250 的初始化流程图

假定所需的波特率为 1200b/s，数据格式为：1 位停止位、7 位数据位、奇校验。

初始化程序如下：

```
START: MOV  DX, 3FBH                ;LCR的地址
       MOV  AL, 80H                 ;开始
       OUT  DX, AL                  ;使 LCR的 D7=1
       MOV  DX, 3F8H                ;DLL的地址
       MOV  AL, 60H                 ;除数为 0060H
       OUT  DX, AL                  ;写除数低 8位
       INC  DX                      ;DLH的地址
       MOV  AL, 0
       OUT  DX, AL                  ;写除数高 8位
       MOV  DX, 3FBH                ;LCR的地址
       MOV  AL, 0AH                 ;1位停止位,7位数据位,奇校验
       OUT  DX, AL                  ;初始化通信控制寄存器
       MOV  DX, 3FCH                ;MCR的地址
       MOV  AL, 03H                 ;使 DTR和 RTS有效
       OUT  DX, AL                  ;初始化 MODEM控制器
       MOV  DX, 3F9H                ;IER的地址
       MOV  AL, 0                   ;禁止所有中断
       OUT  DX, AL                  ;写中断允许寄存器
```

上面的初始化程序是完全按照图 7-46 所示的顺序编写的，即首先写除数寄存器，而要将除数写入，先要使通信控制寄存器的 $D_7=1$，亦即 DLAB=1，然后再写入 16 位的除数 0060H，即十进制数 96。由于加在 $XTAL_1$ 上的时钟频率为 1.8432MHz，故波特率为 1200b/s。

初始化通信控制字为 00001010B。它指定数据为 7 位，停止位为 1 位，奇校验。modem 控制字为 00000011B，使 DTR 和 RTS 均为低电平，即有效状态。最后，将中断允许控制字写入中断允许寄存器。由于中断允许字为 00H，故禁止 4 个中断源可能形成的中断。8250 的中断在硬件上是通过 OUT_2 输出控制的三态门接到 8259 上去的。若允许中断，则一方面要使 OUT_2 输出为低电平，同时，再初始化中断允许寄存器。OUT_2 是由 modem 控制字的 D_3 来控制。只有当 modem 控制字的 $D_3=1$ 时，OUT_2 才为低电平。上述的 modem 控制字为 03H，其 $D_3=0$，故 $OUT_2=1$，这时禁止中断请求输出。

发送数据的程序接在初始化程序之后。若采用查询方式发送数据，且假定要发送的字节数放在 BX 中，要发送的数据顺序存放在以 DATA 为首地址的内存区中，则发送数据的程序段如下：

```
SENDPRG: MOV   DX, 3FDH
         LEA   SI, DATA
WAITTHR: IN    AL, DX
         TEST  AL, 20H              ;检查 THR是否空
         JZ    WAITSE
         PUSH  DX
```

```
        MOV   DX, 3F8H
        LODSB
        OUT   DX, AL                    ;发送一个字节
        POP   DX
        DEC   BX
        JNZ   WAITTHR
```

同样,在初始化后,可以利用查询方式实现数据的接收。下面是接收一个数据的程序段:

```
RECVPRG: MOV   DX, 3FDH
WAITRBR: IN    AL, DX
         TEST  AL, 1EH                  ;检查是否有任何错误产生
         JNZ   ERROR
         TEST  AL, 01H                  ;检查数据准备好否
         JZ    WAITRBR
         MOV   DX, 3F8H
         IN    AL, DX                   ;接收一个字节
         AND   AL, 7FH                  ;只保留低 7位
         …
```

该程序首先测试状态寄存器,看接收的数据是否有错。若有错,就转向错误处理 ERROR;若无错,再看是否已收到一个完整的数据。若是这样,则从 8250 的接收数据寄存器中读出,并取事先约定的 7 位数据,将其放在 AL 中。

下面仍以图 7-45 所示的连接形式为例,说明利用中断方式,通过 8250 实现串行异步通信的过程。为了便于叙述,设想系统以查询方式发送数据,以中断方式接收数据,这时对 8250 的初始化的程序如下:

```
INISIR: MOV  DX, 3FBH
        MOV  AL, 80H
        OUT  DX, AL                     ;置 DLAB= 1
        MOV  DX, 3F8H
        MOV  AL, 0CH
        OUT  DX, AL
        MOV  DX, 3F9H
        MOV  AL, 0                      ;置除数为 000CH,规定波特率为 9600b/s
        OUT  DX, AL
        MOV  DX, 3FBH
        MOV  AL, 0AH                    ;1位停止位,7位数据位,奇校验
        OUT  DX, AL                     ;初始化通信控制寄存器
        MOV  DX, 3FCH
        MOV  AL, 0BH                    ;使 OUT2、DTR和 RTS有效
        OUT  DX, AL                     ;初始化 MODEM寄存器
        MOV  DX, 3F9H
        MOV  AL, 01H                    ;允许接收数据寄存器满产生中断
        OUT  DX, AL                     ;初始化中断允许寄存器
        STI                             ;CPU开中断
```

该程序对 8250 进行初始化,并在初始化完时(假如其他接口初始化在此之前)开中断。接收一个字符的中断服务子程序(接收到一个字符时自动调用此程序)可如下编写:

```
RECVE:     PUSH  AX
           PUSH  BX
           PUSH  DX
           PUSH  DS
           MOV   DX, 3FDH
           IN    AL, DX
           MOV   AH, AL                     ;保存接收状态
           MOV   DX, 3F8H
           IN    AL, DX                     ;读入接收到的数据
           AND   AL, 7FH
           TEST  AH, 1EH                    ;检查有无错误产生
           JZ    SAVEDATA
           MOV   AL, '?'                    ;出错的数据用问号替代
SAVEDATA: MOV    DX, SEG BUFFER
          MOV    DS, DX
          MOV    BX, OFFSET BUFFER
          MOV    [BX], AL
          MOV    AL, 20H                    ;将 EOI 命令发给中断控制器 8259A
          OUT    20H, AL
          POP    DS
          POP    DX
          POP    BX
          POP    AX
          STI
          IRET
```

当 8250 的接收数据寄存器满而产生中断时,此中断请求经过中断控制器 8259A 送给 CPU。CPU 中断响应后,转向上述中断服务子程序。该中断服务子程序首先进行断点和现场的保护;再取回接收状态和接收到的一个字符,并检查接收有无差错,若有错则进行错误处理(本例中对有错的字符用问号替代),无错则将接收到的字符存放在指定的存储单元 BUFFER 中;然后恢复断点,开中断并中断返回。这里特别说明的是,在中断服务子程序结束前,必须给 8259A 一个中断结束命令 EOI(这点在 6.5 节已讲到过),使 8259A 能将中断服务寄存器的状态复位,以便系统又能处理其他低级别的中断。

习　　题

7.1　一般来讲,接口芯片的读写信号应与系统的哪些信号相连?

7.2　试说明 8253 的 6 种工作方式。其时钟信号 CLK 和门控信号 GATE 分别起什么作用?

7.3 8253 可编程计数器有两种启动方式。在软件启动时，要使计数正常进行，GATE 端必须为(　　)电平。如果使硬件启动呢？

7.4 若 8253 芯片的接口地址为 D0D0H～D0D3H，时钟信号频率为 2MHz。现利用计数器 0、1、2 分别产生周期为 10μs 的对称方波及每 1ms 和 1s 产生一个负脉冲，试画出其与系统的电路连接图，并编写包括初始化在内的程序。

7.5 某一计算机应用系统采用 8253 的计数器 0 作为频率发生器，输出频率为 500Hz；用计数器 1 产生 1000Hz 的连续方波信号，输入 8253 的时钟频率为 1.19MHz。试问：初始化时送到计数器 0 和计数器 1 的计数初值分别为多少？计数器 1 工作于什么方式下？

7.6 若要求 8253 用软件产生一次性中断，最好采用哪种工作方式？现用计数器 0 对外部脉冲计数，每计满 10000 个产生一次中断，请写出工作方式控制字及计数初值。

7.7 试比较并行通信与串行通信的特点。

7.8 8255 各端口可以工作在几种方式下？当端口 A 工作在方式 2 时，端口 B 和 C 工作于什么方式下？

7.9 在对 8255 的 C 口进行初始化为按位置位或复位时，写入的端口地址应是(　　)地址。

7.10 某 8255 芯片的地址范围为 A380H～A383H，工作于方式 0，A 口、B 口为输出口，现欲将 PC_4 置“0”，PC_7 置“1”，试编写初始化程序。

7.11 设 8255 的接口地址范围为 03F8H～03FBH，A 组、B 组均工作于方式 0，A 口作为数据输出口，C 口低 4 位作为控制信号输入口，其他端口未使用。试画出该片 8255 与系统的电路连接图，并编写初始化程序。

7.12 已知某 8088 微机系统的 I/O 接口电路框图如图 7-47 所示。试完成以下几点。

(1) 根据图中接线，写出 8255、8253 各端口的地址。

(2) 编写 8255 和 8253 的初始化程序。其中，8253 的 OUT_1 端输出 100Hz 方波，8255 的 A 口为输出，B 口和 C 口为输入。

(3) 为 8255 编写一个 I/O 控制子程序，其功能为：每调用一次，先检测 PC_0 的状态，若 $PC_0=0$，则循环等待；若 $PC_0=1$，可从 PB 口读取当前开关 K 的位置(0～7)，经转换计算从 A 口的 PA_0～PA_7 输出该位置的二进制编码，供 LED 显示。

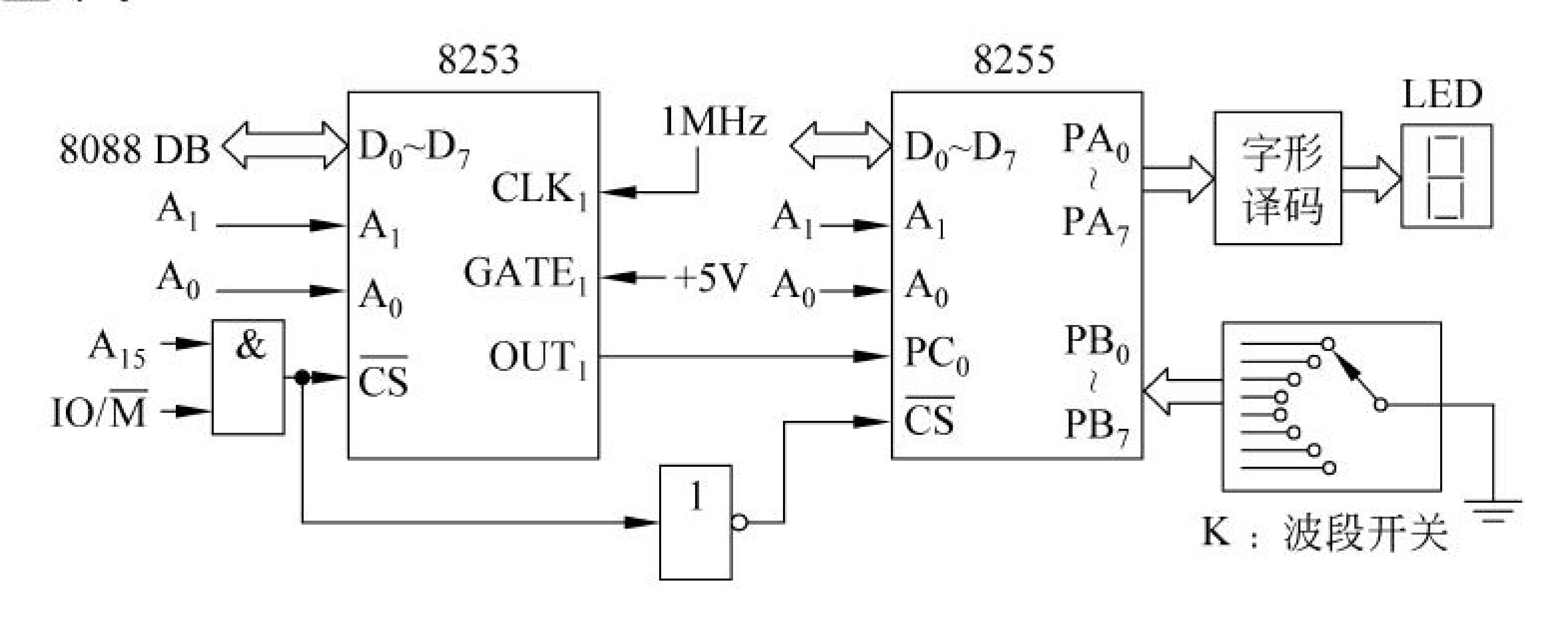


图 7-47　题 7.12 接口电路框图

7.13 试说明串行通信的数据格式。

7.14 串行通信接口芯片 8250 的给定地址为 83A0H～83A7H,试画出其与 8088 系统总线的连接图。采用查询方式由该 8250 发送当前数据段、偏移地址为 BUFFER 的顺序 100 个字节的数据,试编写发送程序。

7.15 题 7.14 中若采用中断方式接收数据,试编写将接收到的数据放在数据段 DATA 单元的中断服务子程序。



第8章 模拟量的输入输出

引言：

“家庭安全防盗系统”中，利用“监测装置”来监控每个窗户是否有异常。这里的监测装置实际上是一类传感器。在第6章和第7章中，我们都假设监测传感器输出的信号是数字信号。事实上，传感器的性质决定了其直接输出的信号通常都是连续变化的模拟信号。

在工业生产中，需要测量和控制的对象往往是连续变化的物理量，如温度、压力、流量、位移等。为了利用计算机实现对工业生产过程的自动监测和控制，首先必须要能够将生产过程中监测设备输出的连续变化的模拟量转变为计算机能够识别和接受的数字量。其次，还要能够将计算机发出的控制命令转换为相应的模拟信号，去驱动模拟调节执行机构。这样两个过程，就需要模拟量的输入和输出通道来完成。因此，模拟量输入输出通道是实现工业过程控制的重要组成部分。通过本章的学习，读者应能够对工业闭环控制系统的整体结构有基本的了解，并能够进行数据采集系统的简单软、硬件系统的设计。

教学目的：

(1) 了解模拟量输入输出通道及其各主要部件的功能；
(2) 理解D/A转换器的基本工作原理及DAC0832芯片的应用；
(3) 了解A/D转换器的基本工作原理；
(4) 掌握ADC0809芯片与系统的连接方法及数据采集程序的设计。

8.1 模拟量的输入输出通道

模拟量输入输出通道的结构如图8-1所示。下面分别介绍输入输出通道中各环节的作用。

8.1.1 模拟量输入通道

典型的模拟量输入通道由以下几部分组成。

1. 传感器

传感器(Transducer)是用于将工业生产现场的某些非电物理量转换为电量(电流、电

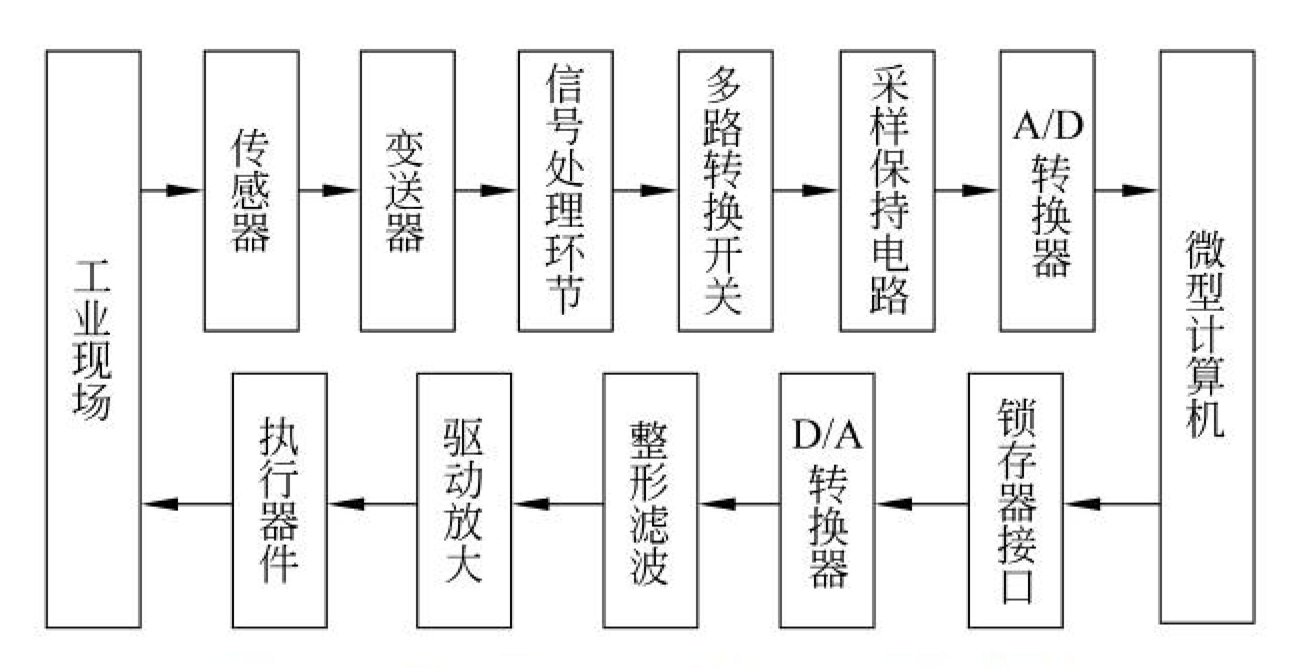


图 8-1 模拟量的输入和输出通道结构图

压)的器件。例如热电偶能够将温度这个物理量转换成几毫伏或几十毫伏的电压信号,所以可用它作为温度传感器;而压力传感器可以把物理量压力的变化转换为电信号;等等。“家庭安全防盗系统”中的监测装置就是一类传感器,可以是压力式传感器、开关式传感器、红外传感器等。不同的监测传感器,其输出信号的类型、格式等都会不同,由此也会使后续的控制方式有所不同。

这里所说的传感器是传统意义上的、仅具备“将物理量转换为电信号”功能的传感器。随着技术的发展,现代许多新型传感器的功能已越来越强大,其内部不仅集成了以下介绍的变送器,还包括信号处理系统,甚至 A/D 转换器,从而使传感器的输出直接为数字信号。

2. 变送器

一般来讲,传感器输出的电信号都比较微弱,有些传感器的输出甚至是电阻值、电容值等非电量。为了易于与信号处理环节衔接,就需要将这些微弱电信号及电阻值等非电量转换成一种统一的电信号,变送器就是实现这一功能的器件。它将传感器的输出信号转换成 0～10mA、4～20mA 的统一电流信号或者 0～5V 等的电压信号。

3. 信号处理环节

信号处理环节主要包括信号的放大及干扰信号的去除。它将变送器输出的信号进行放大或处理成与 A/D 转换器所要求的输入相适应的电压水平。另外,传感器通常都安装在现场,环境比较恶劣,其输出常叠加有高频干扰信号。因此,信号处理环节通常是低通滤波电路,如 RC 滤波器,或由运算放大器构成的有源滤波电路等。

4. 多路模拟开关

在生产过程中,要监测或控制的模拟量往往不止一个,尤其是数据采集系统中,需要采集的模拟量一般比较多,而且不少模拟量是缓慢变化的信号。对这类模拟信号的采集,可采用多路模拟开关(Multiplexer),使多个模拟信号共用一个 A/D 转换器进行采样和转换,以降低成本。

5. 采样保持电路

在数据采样期间,保持输入信号不变的电路称为采样保持电路(Sample Holder)。由

于输入模拟信号是连续变化的，而 A/D 转换器完成一次转换需要一定的时间，这段时间称为转换时间。不同的 A/D 变换芯片，其转换时间不同。对变化较快的模拟输入信号，如果不在转换期间保持输入信号不变，就可能引起转换误差。A/D 转换芯片的转换时间越长，对同样频率模拟信号的转换精度的影响就越大。所以，在 A/D 转换器前面要增一级采样保持电路，以保证在转换过程中输入信号保持在其采样时的值不变。

6. A/D 转换器

A/D 转换器(Analog to Digital)是模拟量输入通道的中心环节，它的作用是将输入的模拟信号转换成计算机能够识别的数字信号，以便计算机进行分析和处理。

8.1.2 模拟量输出通道

计算机的输出信号是数字信号，而有的控制元件或执行机构要求提供模拟的输入电流或电压信号，这就需要将计算机输出的数字量转换为模拟量，这个过程的实现由模拟量的输出通道来完成。输出通道的核心部件是数/模(Digital to Analog，D/A)转换器。由于将数字量转换为模拟量同样需要一定的转换时间，也就要求在整个转换过程中待转换的数字量要保持不变。而计算机的运行速度很快，其输出的数据在数据总线上稳定的时间很短。因此，在计算机与 D/A 转换器之间必须加一级锁存器以保持数字量的稳定。D/A 转换器的输出端一般还要加上低通滤波器，以平滑输出波形。另外，为了能够驱动执行器件，还需要将输出的小功率的模拟量加以放大。

8.2 D/A 转换器

8.2.1 D/A 转换器的基本原理及技术指标

1. D/A 转换器的基本工作原理

D/A 转换器的作用是将数字量转换为相应的模拟量。数字量由二进制位组成，每个二进制位的权为 2^i，要把数字量转换为相应的模拟量电压(多数情况需要转换后的模拟信号以电压的形式输出)，需要先把数字量的每一位上的代码按权转换成对应的模拟电流，再把模拟电流相加，最后由运算放大器将其转变成模拟电压。将数字量转换成对应模拟电流的工作由 D/A 转换器来完成。

典型的 D/A 转换器芯片通常由模拟开关、电阻网络以及运算放大器等组成。其框图如图 8-2 所示。

电阻网络是 D/A 转换器的核心部件。其结构有权电阻网络和 R-2R T 形电阻网络两种主要网络形式。

下面首先介绍一下运算放大器的原理，然后从中引申出 D/A 转换器的工作原理。

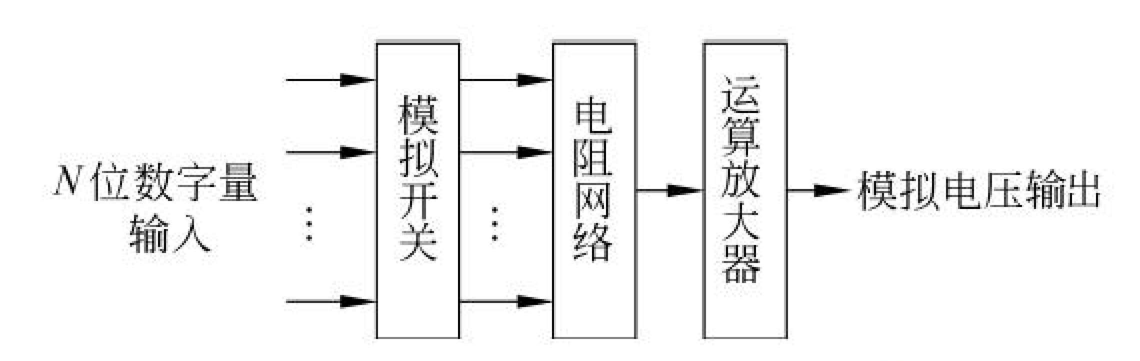


图 8-2 D/A 转换器结构示意图

众所周知，运算放大器具有如下特点。

(1) 开环放大倍数很高(一般为几千到几十万)，因此所需输入电压很小。

(2) 输入阻抗非常大，所以其输入电流很小。

(3) 输出阻抗非常小，使运算放大器的负载能力很强。

一个简单的运算放大器电路如图 8-3 所示。

对运算放大器来说，其输出电压 V_o 与输入电压 V_i 之间有如下关系

$$V_o = -\frac{R_f}{R_i}V_i \tag{8.1}$$

式(8.1)中，R_f 为运算放大器的反馈电阻；R_i 为输入电阻。

若输入端有 n 个支路，如图 8-4 所示，则输入与输出的关系可表示为

$$V_o = -R_f\sum_{j=1}^{n}\frac{1}{R_j}V_i \tag{8.2}$$

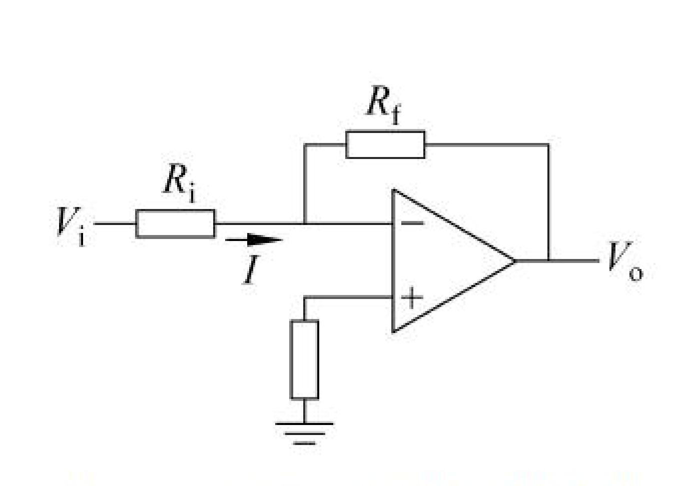


图 8-3 基本运算放大器电路

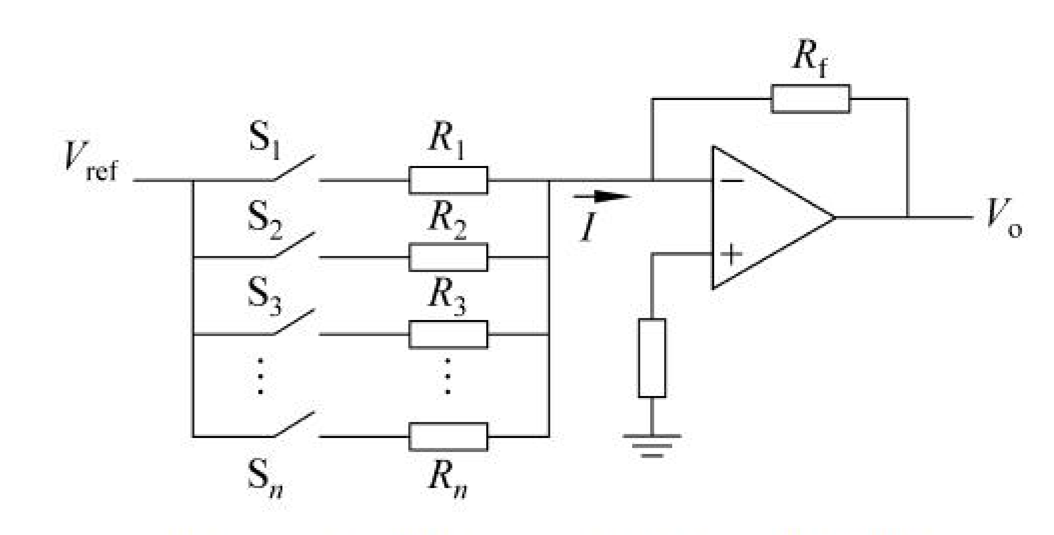


图 8-4 多路输入的运算放大器电路

如果使各支路上的输入电阻 R_1、R_2、…、R_n 分别等于 2^1R、2^2R、…、2^nR，即每一位电阻值都具有权值 2^j(j 为该电阻所在的位数)，且由一个开关 S_j 来控制，当 S_j 合上时 $S_j=1$，S_j 断开时 $S_j=0$，并令 $V_{ref}=\frac{R_f}{R}V_i$，则可得出输出电压 V_o 和输入的关系为

$$V_o = -\sum_{j=1}^{n}\frac{1}{2^j}S_jV_{ref} \tag{8.3}$$

通过式(8.3)可以看出：

(1) 当所有开关 S_j 断开时，$V_o=0$；

(2) 当所有开关 S_j 闭合时，输出电压 V_o 为最大，即 $V_o=-\frac{2^j-1}{2^j}V_{ref}$。

如果用二进制编码来控制图 8-4 中每一路的 S_j，当第 i 路的二进制码为 1 时，使第 j 位的 S_j 闭合；第 j 路的二进制码为 0 时，使对应的 S_j 断开，则数字量的变化就转换成了模拟量的变化。这就是 D/A 转换的基本原理。

D/A 转换器的转换精度与基准电压 V_{ref} 和权电阻的精度以及数字量的位数 j 有关。显然,位数越多,转换精度就越高,但同时所需的权电阻的种类就越多。由于在集成电路中制造高阻值的精密电阻比较困难,因此常用 R-2R T 形电阻网络来代替权电阻网络,如图 8-5 所示。这是一个简化了的 T 形电阻网络原理图。它只由两种阻值 R 和 $2R$ 组成,用集成工艺生产较为容易,精度也容易保证,因此得到比较广泛的应用。式(8.4)为 R-2R T 形电阻网络的输出和输入电压的关系表达式。

$$V_o = \frac{-D}{2^j} \times \frac{R_f}{R} \times V_{ref} \tag{8.4}$$

式中,D 为输入的数字量;j 为数字量的位数。

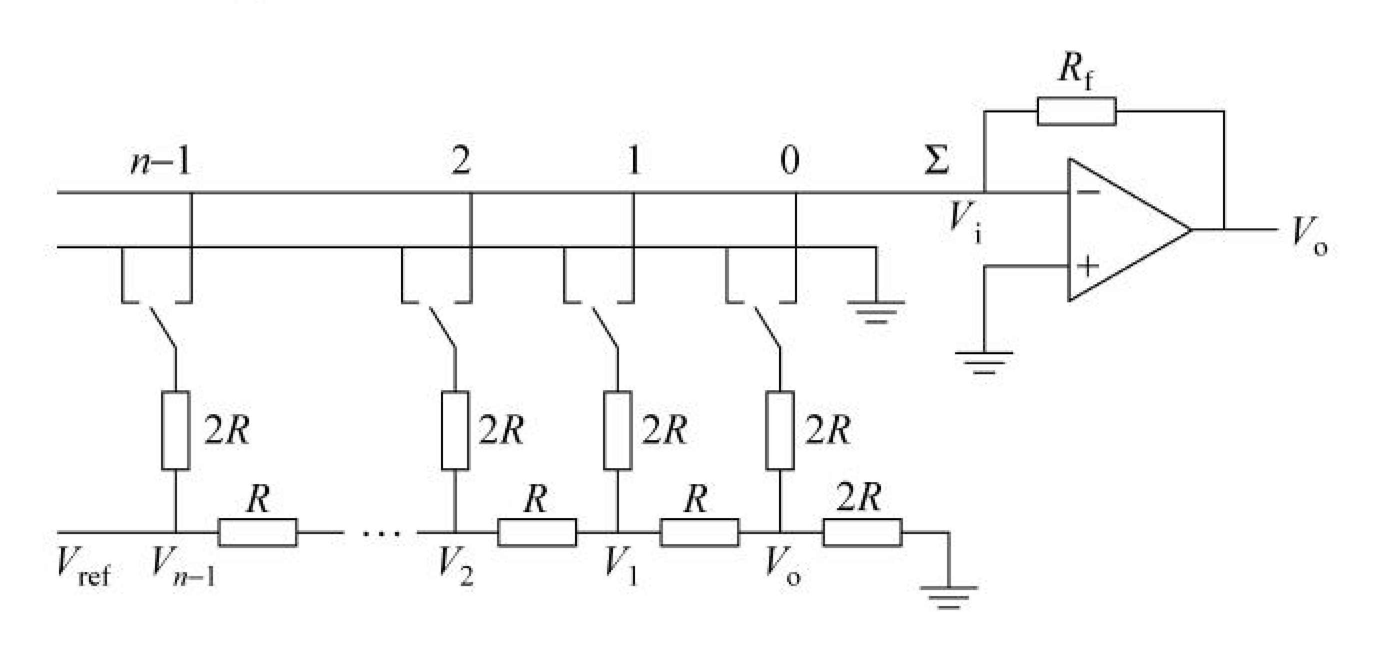


图 8-5　R-2R T 形电阻网络

由式(8.4)可知,输出电压 V_o 正比于输入数字量 D,而幅度大小由 V_{ref} 和 R_f/R 的比值决定。若使 $R_f/R=1$,并且输入为 8 位的数字量,则上式可简化为式(8.5),即 8 位 D/A 转换器的输出电压与数字量的关系式。

$$V_o = \frac{-D}{256} \times V_{ref} \tag{8.5}$$

电阻网络是构成 D/A 转换器的主要部件,但在具体电路中还需要一些其他部件。一个实际的 D/A 转换器原理框图如图 8-6 所示。

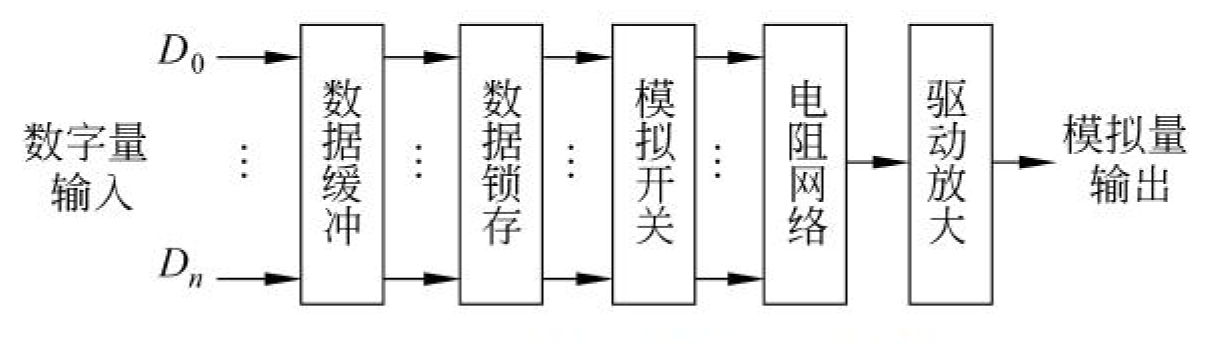


图 8-6　D/A 转换器原理框图

首先将待转换的数字量 $D_0 \sim D_n$ 通过数据缓冲器送至数据锁存器,以确保在整个转换过程中数字量的稳定(仅在一次转换过程结束后,才允许将新的数字量存入)。锁存器的输出接到多路模拟开关,使数据信号的高低电平转变成相应的开关状态。各位模拟开关输出的电流通过电阻网络进行加权,合成一个与输入数字量等效的模拟电流信号,再经过驱动放大电路,形成模拟量的输出。

有时,需要 D/A 转换器输出电压信号,对这种情况,可在其输出端接一个运算放大器,将电流信号转换为电压信号输出。

D/A 转换器的输出形式有电压、电流两大类。电压输出型的 D/A 转换器的输出电压一般为 0～5V 或 0～10V，它相当于一个电压源，内阻较小，可带动较大的负载。而电流输出型的则相当于一个电流源，内阻较大，与之匹配的负载电阻不能太大。

2. D/A 转换器的主要技术指标

1）分辨率

分辨率(Resolution)是 D/A 转换器对数字输入量变化的敏感程度的度量。它表示输入每变化一个最低有效位使输出变化的程度，可用数字量的位数来表示，如 8 位、10 位等，也可定义为输入数字量等于 1 时的电压值与输入数字量等于最大值时的满度模拟值之比。例如，对一个 n 位的 D/A 转换器，若其满度电压值为 V，其最低有效位对应的电压值就为 $V/(2^n-1)$，则该 D/A 转换器的分辨率等于 $1/(2^n-1)$。如果用百分比表示，则为 $[1/(2^n-1)]\times 100\%$。

2）转换精度

转换精度表示由于 D/A 转换器的引入而使其输出和输入之间产生的误差，可用绝对转换精度或相对转换精度来表示。

绝对转换精度是指实际的输出值与理论值之间的差距。它与 D/A 转换器参考电压的精度、权电阻的精度等有关。

相对转换精度是绝对转换精度与满量程输出之比再乘以 100%，是常用的描述输出电压接近理想值程度的物理量，更具有实用性。例如，一个 D/A 转换器的绝对转换精度为±0.05V，若输出满刻度值为 5V，则其相对转换精度为±1%。

与 D/A 转换器转换精度有关的指标还有以下几点。

(1) 非线性误差——在满刻度范围内，偏移理想的转换特性的最大值。

(2) 温度系数误差——在允许范围内，温度每变化 1℃所引起的输出变化。

(3) 电源波动误差——由于电源的波动引起的输出变化。

(4) 运算放大器误差——与 D/A 变化器相连的运算放大器带来的误差。

需要注意的是，由于不可能用有限位数的数字量来表示连续的模拟量，所以由位数产生的转换误差是不能消除的，是系统固有的。为了尽量减小分辨率造成的转换误差，在系统设计时，应这样来选择 D/A 转换器的位数，使其最低有效位的变化所引起的误差远远小于 D/A 芯片的总误差。

3）转换时间

转换时间是指当输入数字量满刻度变化(如全 0 到全 1)时，从数字量输入到输出模拟量达到与终值相差±1/2 LSB(最低有效位)相当的模拟量值所需的时间。它表征了一个 D/A 转换器芯片的转换速率。

4）线性误差

在 D/A 转换时，若数据连续转换，则输出的模拟量应该是线性的，即在理想情况下，D/A 转换器的输入输出曲线是一条直线，但实际的输出特性曲线与理想的曲线之间存在一定的误差。将实际输出特性偏离理想转换特性的最大值称为线性误差。通常用这个最大差值折合成的数字量来表示。

例如一个 D/A 转换器的线性误差小于 1/2 LSB,表示用它进行 D/A 变换时,其输出模拟量与理想值之差最大不会超过 1/2 LSB 的输入量产生的输出值。

5) 动态范围

D/A 转换器的动态范围是指最大和最小输出值范围,一般决定于参考电压 V_{ref} 的高低。参考电压高,动态范围就大。整个 D/A 转换电路的动态范围除与 V_{ref} 有关外,还与输出电路的运算放大器的级数及连接方法有关。适当地选择输出电路可在一定程度上增加转换电路的动态范围。

8.2.2 典型 D/A 转换器芯片 DAC0832

D/A 转换器的种类繁多,在目前常用的 D/A 芯片中,从数码位数上看,有 8 位、10 位、16 位等;从输出形式上看,有电流输出和电压输出;从内部结构上,又可分为含数据输入寄存器和不含数据输入寄存器两类。对内部不含数据输入寄存器的芯片,亦即不具备数据锁存能力的芯片,不能直接与系统总线连接。因为对 D/A 转换器来讲,当有数字量输入时,其输出端随之就会有模拟电流或电压信号建立;而当输入端数字量消失时,输出模拟量也随之消失。另外,为实现对某个对象的控制,要求输出模拟量要能够保持一段时间。在微机系统中,D/A 转换器的输入数据来自 CPU,8088 CPU 在执行输出指令时,数据在数据总线上只能维持两个时钟周期,这使得转换后的模拟量在输出端保持时间太短,无法满足实际控制系统的要求。所以,这类芯片(如 AD7520、AD7521 等)在与 CPU 连接时,要求在其与 CPU 之间增加数据锁存器(如 74LS273)。而内部已包含数据输入寄存器的 D/A 转换器芯片可直接与系统总线相连,常见的有 DAC0832、AD7524 等。

尽管 D/A 转换器的型号很多,但它们的基本工作原理和功能都是一致的。下面就以较常用的 DAC0832 为例,来说明数/模转换器与 CPU 的连接方法及其应用。

1. 引线及内部结构

DAC0832 是一个 8 位的数/模转换芯片,内部包含一个 T 形电阻网络,输出为差动电流信号。要想得到模拟电压输出,必须外接运算放大器。其外部引线图和内部结构示意图分别如图 8-7 和图 8-8 所示。

图 8-7 中各引脚的定义如下。

(1) $D_0 \sim D_7$:8 位数据输入端。

(2) $\overline{CS}$:片选信号,低电平有效。

(3) ILE:输入寄存器选通信号,它与 $\overline{CS}$、$\overline{WR_1}$ 一起将要转换的数据送入输入寄存器。

(4) $\overline{WR_1}$:输入寄存器的写入控制,低电平有效。

(5) $\overline{WR_2}$:数据变换(DAC)寄存器写入控制,低电平有效。

(6) $\overline{XFER}$:传送控制信号,低电平有效。它与 $\overline{WR_2}$ 一起把输入寄存器的数据装入数据变换寄存器。

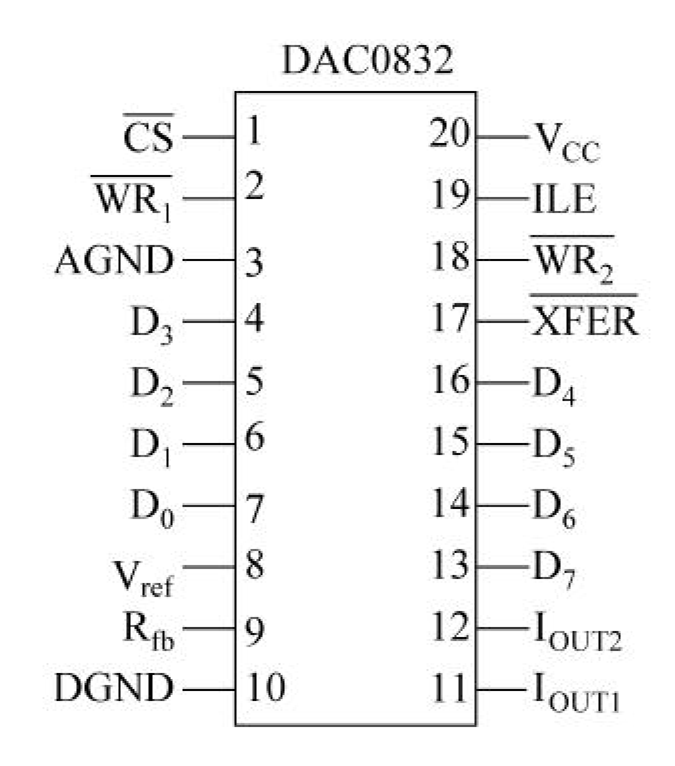


图 8-7 DAC0832 的外部引线图

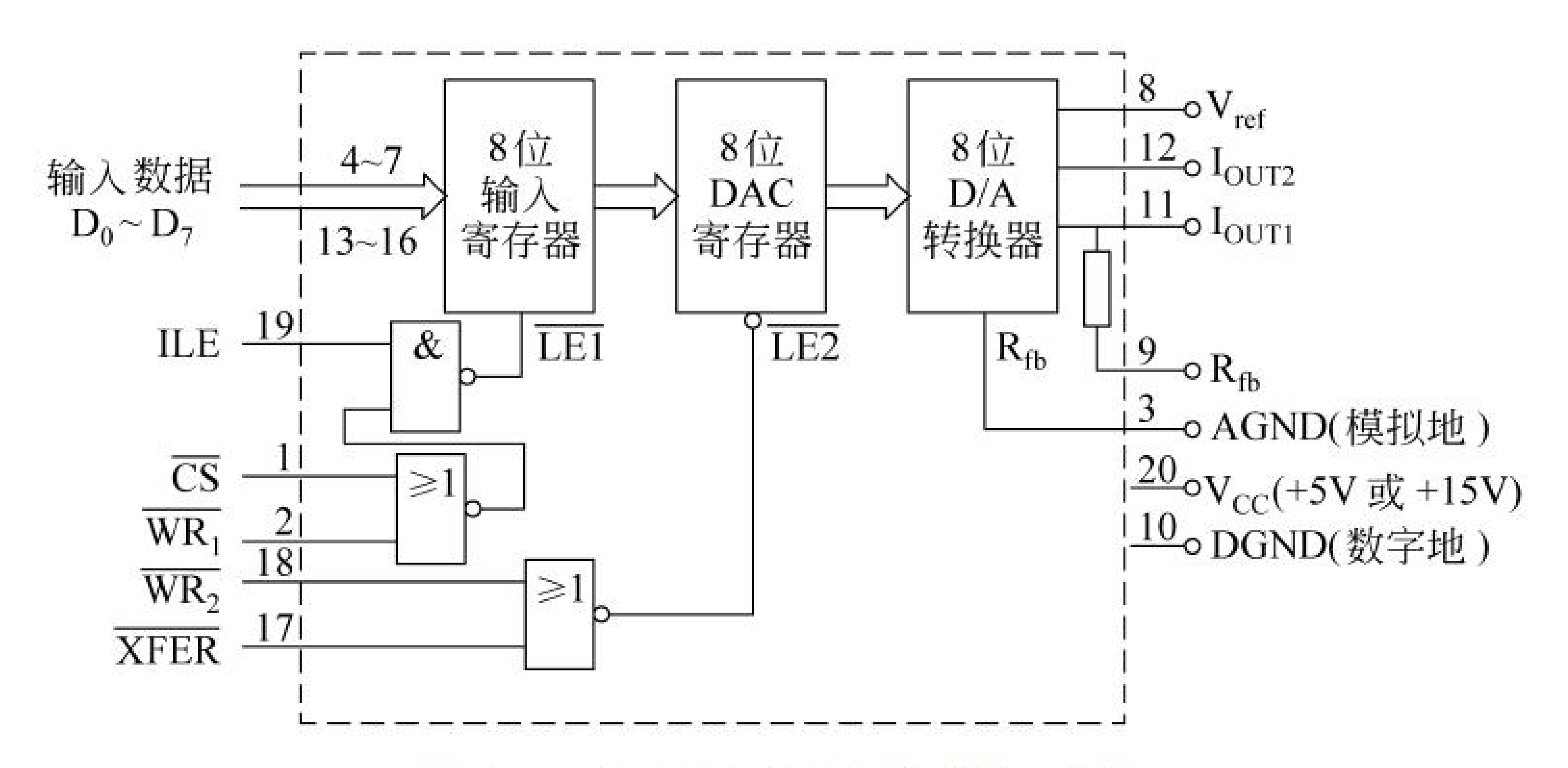


图 8-8　DAC0832 的内部结构示意图

(7) I_{OUT1}：模拟电流输出端，当 DAC 寄存器中内容为 0FFH 时，I_{OUT1} 电流最大；当 DAC 寄存器中内容为 00H 时，I_{OUT1} 电流最小。

(8) I_{OUT2}：模拟电流输出端。DAC0832 为差动电流输出，一般情况下 $I_{OUT1}+I_{OUT2}=$ 常数。

(9) R_{fb}：反馈电阻引出端，接运算放大器的输出。

(10) V_{ref}：参考电压输入端，要求其电压值要相当稳定，一般为－10～＋10V。

(11) V_{CC}：芯片的电源电压，可为＋5V 或＋15V。

(12) AGND：模拟信号地。

(13) DGND：数字信号地。

2. 主要技术指标

DAC0832 的主要技术指标如下。

(1) 分辨率：8 位。

(2) 线性误差：(0.05%～0.2%)FSR(满刻度)。

(3) 转换时间：1μs。

(4) 功耗：20mW。

3. 工作方式及线路连接

从图 8-8 可以看出，DAC0832 的内部包括两级锁存器：第一级是 8 位的数据输入寄存器，由控制信号 ILE、$\overline{CS}$和$\overline{WR_1}$控制；第二级是 8 位的 DAC 寄存器，由控制信号$\overline{WR_2}$和$\overline{XFER}$控制。根据这两个锁存器使用方法的不同，DAC0832 有 3 种工作方式。

1) 单缓冲工作方式

单缓冲工作方式是使输入寄存器或 DAC 寄存器中的任意一个工作在直通状态，而另一个工作在受控锁存状态。例如，要想使输入寄存器受控、DAC 寄存器直通，则可将$\overline{WR_2}$和$\overline{XFEB}$接数字地，ILE 接＋5V。此时，将$\overline{CS}$接端口地址译码器输出，$\overline{WR_1}$接$\overline{IOW}$信号，则当 CPU 向输入寄存器的端口地址发出写命令时(即执行指令 OUT<输入寄存器端口地址>，<要转换的数据>)，数据就写入输入寄存器，因为 DAC 寄存器为直通状态，

所以写入到数据寄存器的数据立刻进行数模变换。其电路连接如图 8-9 所示。

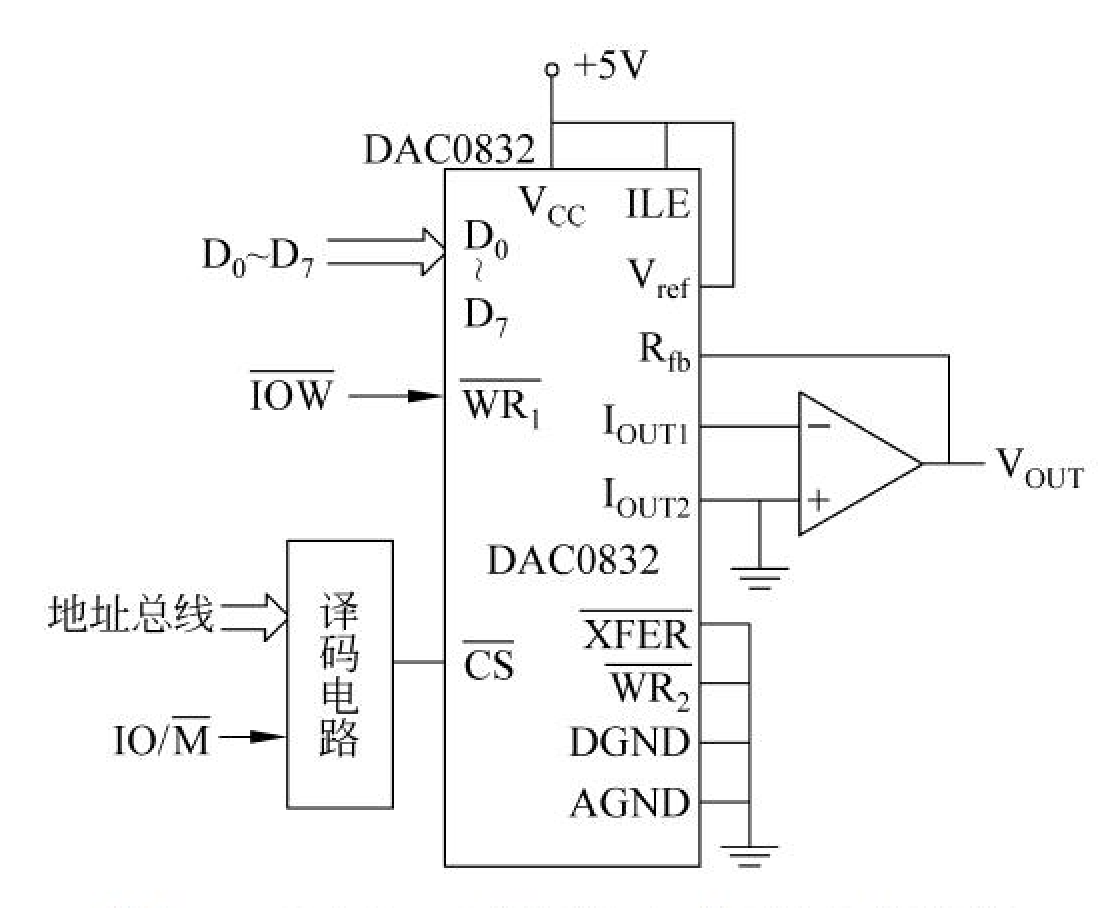


图 8-9　DAC0832 单缓冲方式下的电路连接

在只有单路模拟量输出通道或虽有多路模出通道但不要求同时刷新模拟输出时，可采用这种方式。此种工作方式只用一条输出指令即可完成转换。

【例 8-1】 利用 DAC0832 实现 D/A 变换。0832 工作在单缓冲方式。设 0832 端口地址为 PORT，待转换数据在 DATA 单元中。完成 D/A 转换的程序段如下：

```
MOV   AL, DATA                          ;要转换的数据送 AL
MOV   DX, PORT                          ;0832 的端口地址送 DX
OUT   DX, AL                            ;将数字量送 D/A 转换器进行转换
HLT
```

2）双缓冲工作方式

在这种工作方式下，CPU 要对 0832 进行两步写操作：①将数据写入输入寄存器；②将输入寄存器的内容写入 DAC 寄存器。具体过程为：当 ILE＝1，$\overline{CS}=\overline{WR_1}=0$ 时，待转换的数据被写入输入寄存器；随后，$\overline{WR_1}$由低变高，数据出现在输入寄存器的输出端。在整个$\overline{WR_1}$为高电平期间，输入寄存器的输出端将不再随其输入端的变化而变化，从而保证了在数模转换时数据稳定不变。

锁存在输入寄存器中的数据此时并不能进入 DAC 寄存器，只有当$\overline{XFER}=\overline{WR_2}=0$时，数据才能写入 DAC 寄存器，并同时启动变换。双缓冲的工作时序如图 8-10 所示。其连接方法是：ILE 固定接＋5V，$\overline{WR_1}$、$\overline{WR_2}$均接到$\overline{IOW}$，而$\overline{CS}$和$\overline{XFER}$分别接到两个端口的地址译码信号线，即 0832 占用两个端口地址。

双缓冲工作方式的优点是数据接收和启动转换可以异步进行，可以在 D/A 转换的同时接收下一个数据，提高了模/数转换的速率。它还可用于多个通道同时进行 D/A 变换的场合。其外部接线如图 8-11 所示。

由于这种工作方式要求先使数据锁存到输入寄存器，之后再使数据进入 DAC 寄存器进行数模变换，所以，在程序中需要安排两条 OUT 指令。双缓冲方式的程序段如下：

```
MOV   AL, DATA
```

```
MOV  DX, PORT1                          ;输入寄存器端口地址送 DX
OUT  DX, AL                             ;数据送输入寄存器
MOV  DX, PORT2                          ;DAC 寄存器端口地址送 DX
OUT  DX, AL                             ;数据送 DAC 寄存器并启动变换
HLT
```

3）直通工作方式

直通工作方式是将$\overline{CS}$、$\overline{WR_1}$、$\overline{WR_2}$以及$\overline{XFER}$引脚都直接接数字地，ILE 接＋5V，芯片就处于直通状态。此时 0832 就一直处于 D/A 转换状态，即模拟输出端始终跟踪输入端 D_0～D_7 的变化。由于这种工作方式下 0832 不能直接与 8088 CPU 的数据总线相连接，故在实际工程实践中很少采用。

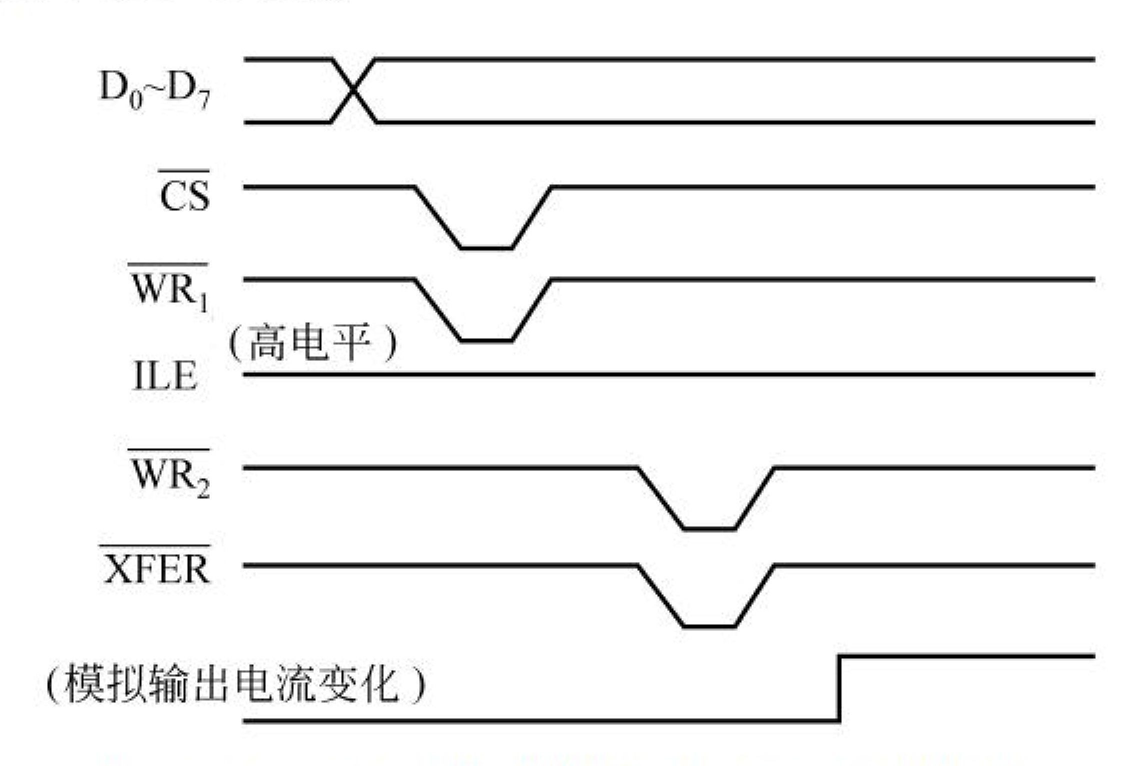


图 8-10　DAC0832 双缓冲方式下工作时序图

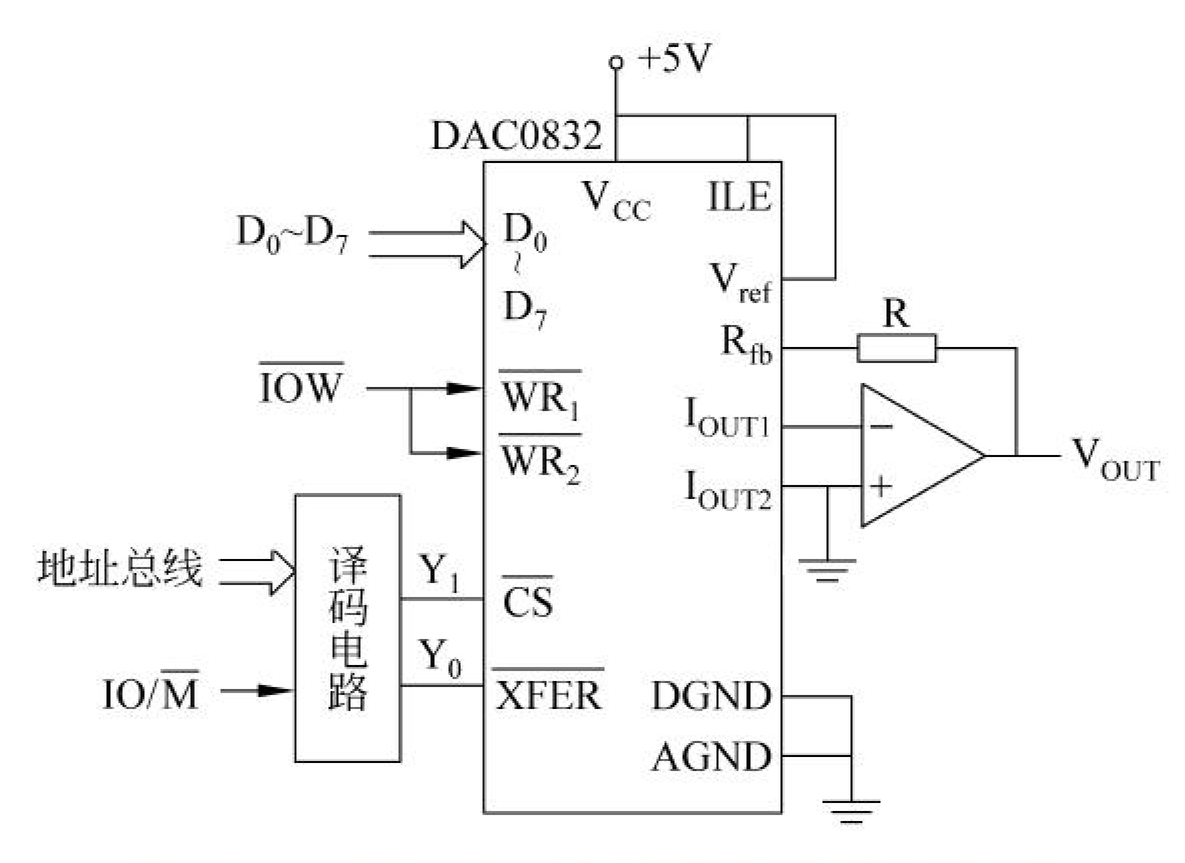


图 8-11　DAC0832 双缓冲方式下的电路连接

8.2.3　D/A 转换器的应用

1. 信号源

由前面的讨论可知，DAC0832 在单缓冲方式下可以直接与系统总线相连，亦即可以

将它看做一个输出端口。每向该端口送一个 8 位数据，其输出端就会有相应的输出电压。可以通过编写程序，利用 D/A 转换器产生各种不同的输出波形，如锯齿波、三角波、方波、正弦波等。

【例 8-2】 根据图 8-12 的电路连接，编写一个输出锯齿波的程序，周期任意。DAC0832 工作在单缓冲方式，端口地址为 0278H。

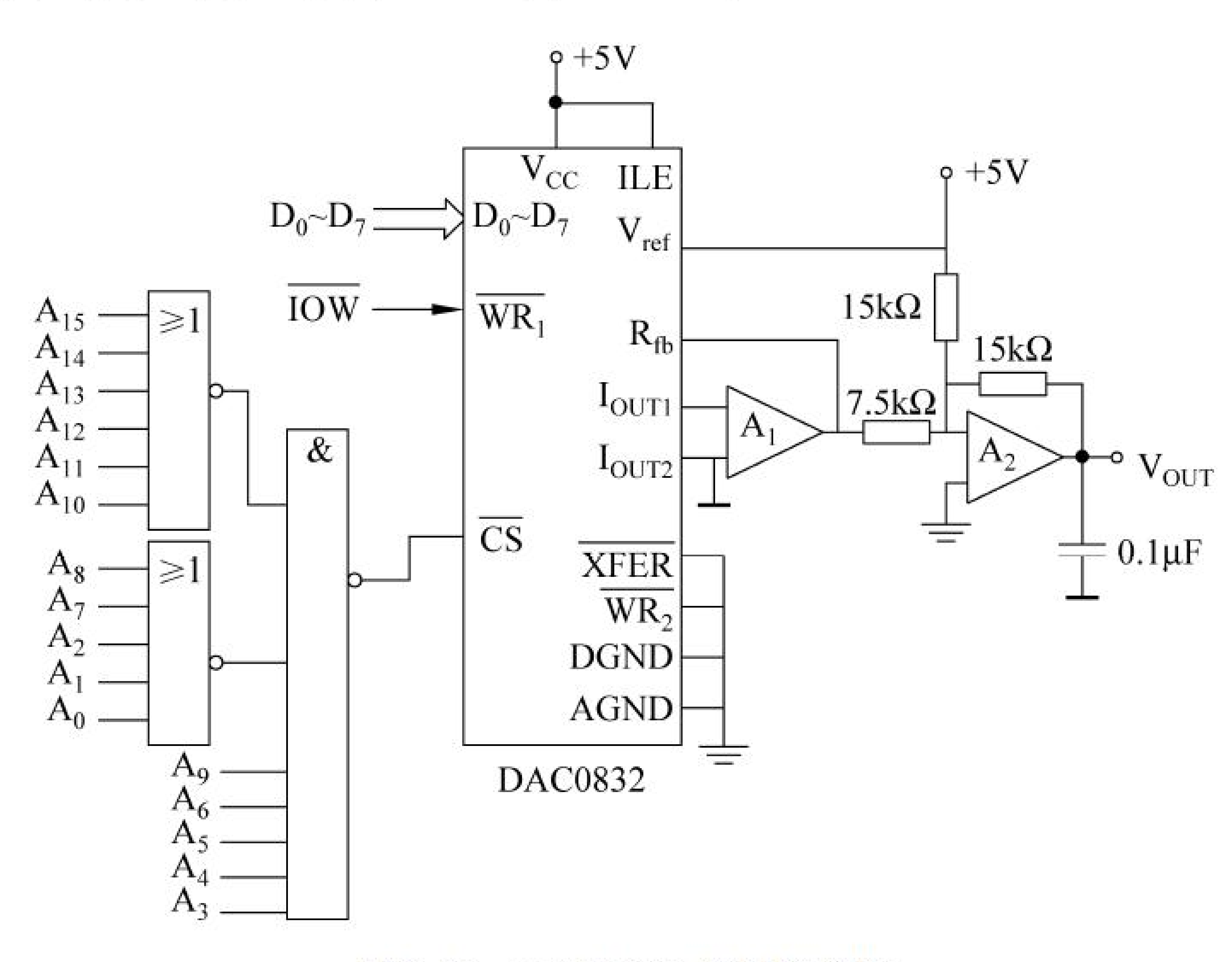


图 8-12　DAC0832 应用连接图

题目分析：

正向锯齿波的规律是电压从最小值开始逐渐上升，上升到最大值时立刻跳变为最小值，如此循环(反向锯齿波正好相反，先从最小值跳变为最大值，然后逐渐下降到最小值)。所以只要从 0 开始往 0832 输入数据，每次加 1，直到最大值 FFH，然后再从 0 开始下一个周期。这个过程循环执行即可在 0832 输出端得到一个正向锯齿波。以下是产生反向锯齿波的程序段，这里使用了一个技巧，用 0 减 1 直接得到最大值 FFH，这样在锯齿波的齿根部可以少做一次判断。

```
        MOV  DX, 0278H                 ;端口地址送 DX
        MOV  AL, 0                     ;初始值送 AL
NEXT:   OUT  DX, AL                    ;输出数字量到 D/A转换器
        DEC  AL                        ;数字量减 1
        JMP  NEXT                      ;循环
```

例 8-2 程序产生的锯齿波不是平滑的波形，而是有 255 个小台阶，通过加滤波电路可以得到较平滑的锯齿波输出，还可以通过软件实现对输出波形周期和幅度的调整。

【例 8-3】 已知 0832 输出电压范围为 0～5V，现希望输出电压为 1～4V、周期任意的正向锯齿波。

题目分析：

考虑到输出波形应能够停止，程序中增加了在有任意键按下时则停止输出的功能。

由题知,已知当输出为5V时,输入数字量为最大值255,则

$$1\text{V 电压对应的数字量} = 1 \times 255/5 = 51 = 33\text{H}$$

$$4\text{V 电压对应的数字量} = 4 \times 255/5 = 204 = \text{CCH}$$

程序设计如下:

```
        MOV   DX, 0278H              ;0832的端口地址送 DX
NEXT1:  MOV   AL, 33H                ;最低输出电压对应的数字量送 AL
NEXT2:  OUT   DX, AL                 ;输出数字量到 0832
        INC   AL                     ;数字量加 1
        CALL  DELAY                  ;调用延时子程序
        CMP   AL, 0CCH               ;到最大值(输出 4V电压)否?
        JNA   NEXT2                  ;若没有到最大值继续输出
        MOV   AH, 1                  ;达到最大输出则判断有无任意键按下
        INT   16H
        JZ    NEXT1                  ;若无任意键按下则重新开始下一个周期
        HLT                          ;有键按下则退出
DELAY   PROC
        MOV   CX, 100                ;延时子程序(延时常数可修改)
DELAY1: LOOP  DELAY1
        RET
DELAY   ENDP
```

例8-3中,不仅实现了波形幅度的调整,通过在延时子程序中设置不同的延时常数还可以实现输出信号周期的调整。

2. 工业控制器

D/A转换器也常用于调速系统和伺服控制系统中的电机转速控制。图8-13给出了一个直流伺服电机的脉宽调制(PWM)转速控制系统。CPU发出的控制信号经锁存器到D/A转换器,转换后的模拟电压通过功率放大器控制直流伺服电动机的转速。速度传感器(如光电编码器等)将检测到的转速通过模拟量的输入通道反馈给微型机,形成闭环控制系统。

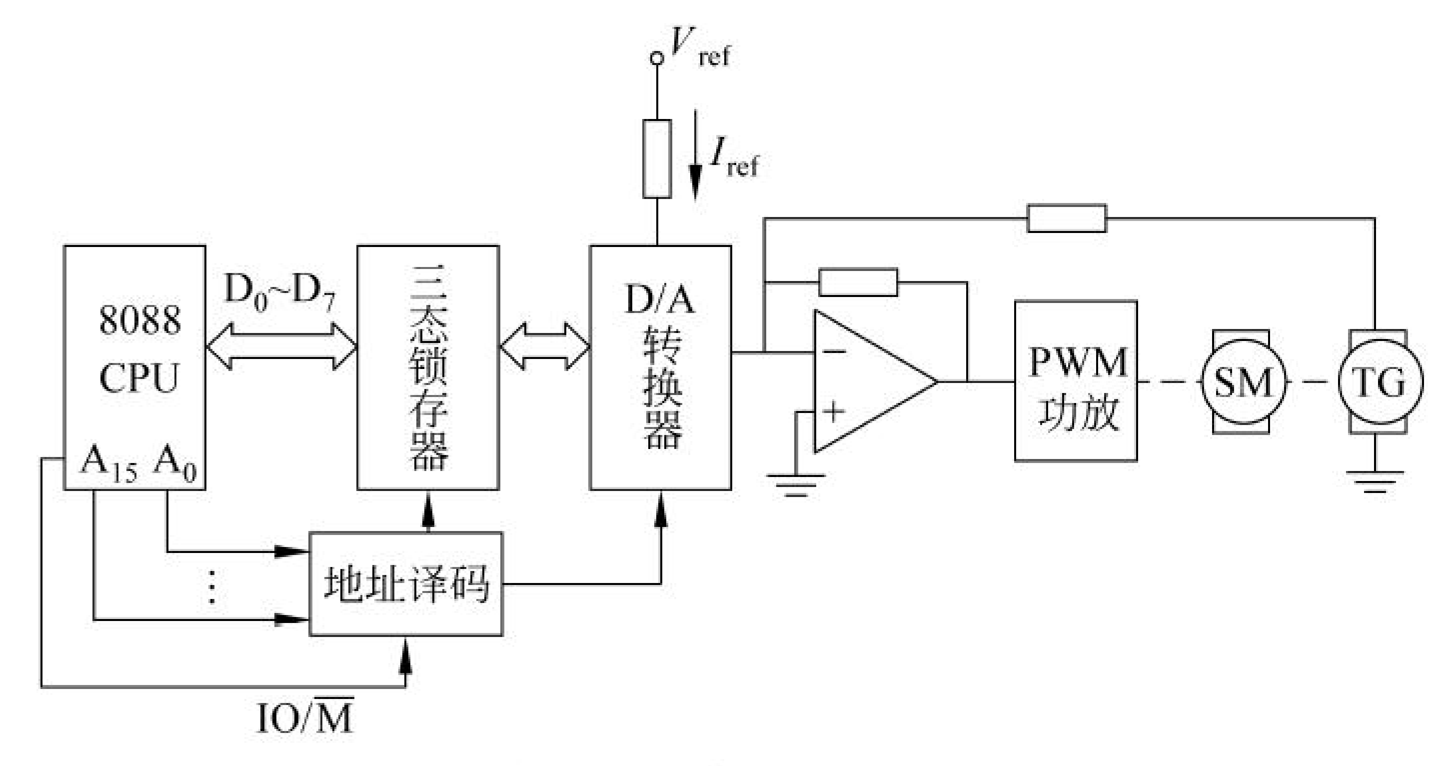


图8-13 D/A转换器在直流电机调速系统中的应用

8.3 A/D 转换器

A/D 转换器是将连续变化的模拟信号转换为数字信号,以便于计算机进行处理。它与 D/A 转换器一样,是微型机应用系统中的一种重要接口,常用于数据采集系统。

A/D 转换器的种类很多,如计数型 A/D 转换器、双积分型 A/D 转换器、逐位反馈型 A/D 转换器等。考虑到精度及变换速度的折中,这里以常用的逐位反馈型(也叫逐位逼近型)A/D 转换器为例,来说明 A/D 转换器的一般工作原理。

8.3.1 A/D 转换器的工作原理及技术指标

1. A/D 转换器的工作原理

图 8-14 为逐位反馈型 A/D 转换器的内部结构,主要由逐次逼近寄存器 SAR、D/A 转换器、电压比较器和一些时序及控制逻辑电路等组成。

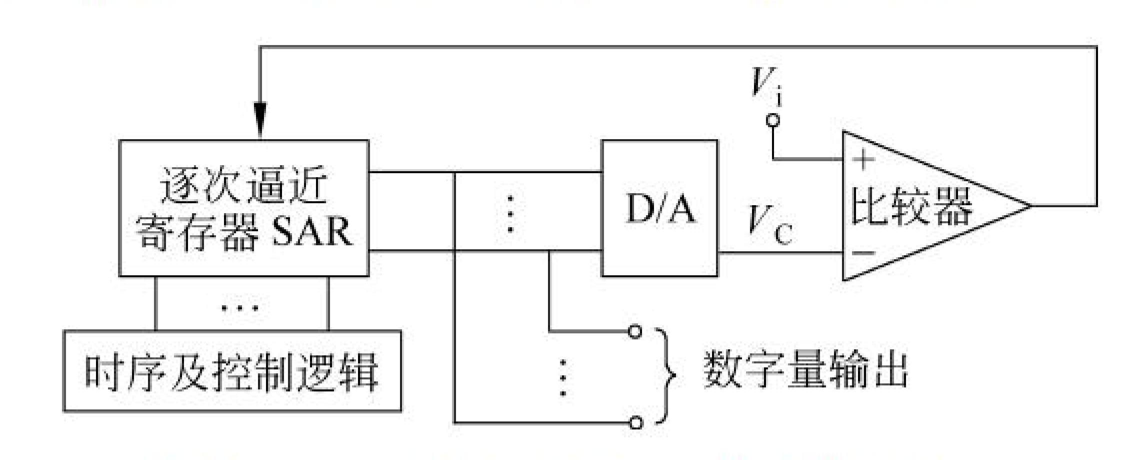


图 8-14 逐位反馈型 A/D 转换器的结构

逐位反馈型 A/D 转换器的工作原理类似于用天平称重。在转换开始前,先将 SAR 寄存器各位清零,然后设其最高位为 1(对 8 位来讲,即为 10000000B)——就像天平称重时先放上一个最重的砝码一样,SAR 中的数字量经 D/A 转换器转换为相应的模拟电压 V_C,并与模拟输入电压 V_i 进行比较,若 $V_i \geqslant V_C$,则 SAR 寄存器中最高位的 1 保留,否则就将最高位清零——若砝码比物体轻就要保留此砝码,否则去掉此砝码。然后再使次高位置 1,进行相同的过程……直到 SAR 的所有位都被确定。转换过程结束后,SAR 寄存器中的二进制码就是 A/D 转换器的输出。

例如,某一个 12 位的 A/D 转换器,如果输入的模拟电压为 0～5V,则输出的对应值就为 0～FFFH,且最低有效位所对应的输出电压为 $5/(2^{12}-1)=1.22\text{mV}$。现设输入模拟电压为 4.5V,其变换过程如下:

位序号	比较表达式		二进制值
D_{11}	$4.5000\text{V}-2^{11}\times 1.22\text{mV}=2\text{V}$	>0	1
D_{10}	$2.0000\text{V}-2^{10}\times 1.22\text{mV}=0.75\text{V}$	>0	1
D_9	$0.7500\text{V}-2^{9}\times 1.22\text{mV}=0.125\text{V}$	>0	1
D_8	$0.1250\text{V}-2^{8}\times 1.22\text{mV}$	<0	0

D_7	$0.1250V-2^7\times1.22mV$	<0	0
D_6	$0.1250V-2^6\times1.22mV=0.046V$	>0	1
D_5	$0.0460V-2^5\times1.22mV=0.0069V$	>0	1
D_4	$0.0069V-2^4\times1.22mV$	<0	0
D_3	$0.0069V-2^3\times1.22mV$	<0	0
D_2	$0.0069V-2^2\times1.22mV=0.0021V$	>0	1
D_1	$0.0021V-2^1\times1.22mV$	<0	0
D_0	$0.0021V-2^0\times1.22mV$	>0	1

这样，就把 4.5V 模拟量转换成了数字量 111001100101B(E65H)。

2. A/D 转换器的主要技术指标

1）精度

A/D 转换器的转换精度由各种因素引起的误差所共同决定。这些误差分别如下。

(1) 量化误差。A/D 转换器的量化误差(也称分辨率)决定于 A/D 转换器的转换特性。例如，一个 3 位的 A/D 转换器的转换特性如图 8-15 所示。当模拟量的值在 0～0.5V 范围变化时，数字量输出为 000B；在 0.5～1.5V 范围变化时，数字量输出为 001B。这样在给定数字量情况下，实际模拟量与理论模拟量之差最大为±0.5V。这种误差是由转换特性造成的，是一种原理性误差，也是无法消除的误差。从图中可以发现，数字量的每个变化间隔为 1V，就是说模拟量在 1V 内的变化，不会使数字量发生变化。这个间隔称为量化间隔(也称为当量)，用 Δ 表示，其定义为

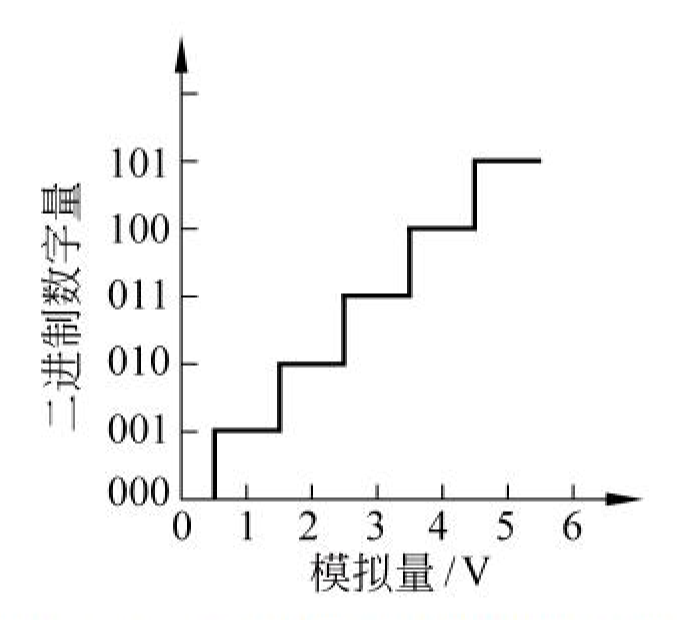


图 8-15　A/D 转换器的转换特性

$$\Delta=\frac{\text{输入满度电压值}}{\text{A/D 转换器的最大数字量输出}} \tag{8.6}$$

对输出为 n 位的 A/D 转换器，其量化间隔 Δ 可表示为

$$\Delta=\frac{V_{\max}}{2^n-1} \tag{8.7}$$

例如，对上例中的 12 位 A/D 转换器，若最大输入模拟电压为 5V，则其量化间隔 Δ 为

$$\Delta=\frac{5V}{4095}\approx1.22mV \tag{8.8}$$

而量化误差用绝对误差就可表示为

$$\text{量化误差}=\frac{1}{2}\times\text{量化间隔}=\frac{V_{\max}}{2\times(2^n-1)} \tag{8.9}$$

也可用(1/2)LSB 来表示。

因此，一旦 A/D 转换器的位数确定，其量化误差也就确定了。

(2) 非线性误差。A/D 转换器的非线性误差是指在整个变换量程范围内，数字量所对应的模拟输入信号的实际值与理论值之间的最大差值。理论上 A/D 变换曲线应该是

一条直线，即模拟输入与数字量输出之间应该是线性关系。但实际上它们两者的关系并非呈线性。所谓非线性误差就是由于二者关系的非线性而偏离理想直线的最大值，常用多少 LSB 来表示。

（3）其他误差。影响 A/D 转换器转换精度的因素还有电源波动引起的误差、温度漂移误差、零点漂移误差、参考电源误差等。

2）转换时间

转换时间是指完成一次 A/D 变换所需要的时间，即从发出启动转换命令信号到转换结束信号之间有效的时间间隔。转换时间的倒数称为转换速率（频率）。例如 AD574KD 的转换时间为 35μs，其转换速率为 28.57kHz。

3）输入动态范围

输入动态范围也称量程，指能够转换的模拟输入电压的变化范围。A/D 转换器的模拟电压输入分为单极性和双极性两种。

（1）单极性：动态范围为 0～＋5V、0～＋10V 或 0～＋20V。

（2）双极性：动态范围为－5～＋5V 或－10～＋10V。

8.3.2 典型 A/D 转换器芯片 ADC0809

A/D 转换器芯片的种类很多。下面以较为常用的 A/D 转换器 ADC0809 为例，介绍 A/D 芯片与微型机系统的连接及应用。

ADC0809 是逐位逼近型 8 位单片 A/D 转换芯片。片内含 8 路模拟开关，可允许 8 路模拟量输入。片内带有三态输出缓冲器，因此可直接与系统总线相连。它的转换精度和转换时间都不是很高，但其性能价格比有较明显的优势，是目前应用较为广泛的芯片之一。

1. ADC0809 的引线及内部结构

1）ADC0809 的外部引线

ADC0809 的外部引线如图 8-16 所示，共有 28 根引脚，其含义如下。

（1）D_0～D_7：输出数据线。

（2）IN_0～IN_7：8 路模拟电压输入端，可连接 8 路模拟量输入。

（3）ADDA、ADDB、ADDC：通道地址选择，用于选择 8 路中的一路输入。ADDA 为最低位，ADDC 为最高位。

（4）START：启动信号输入端，下降沿有效。在启动信号的下降沿启动变换。

（5）ALE：通道地址锁存信号，用来锁存 ADDA～ADDC 端的地址输入，上升沿有效。

（6）EOC：变换结束状态信号。当该引脚输出低电平时表示正在变换，输出高电平则表示一次变换已结束。

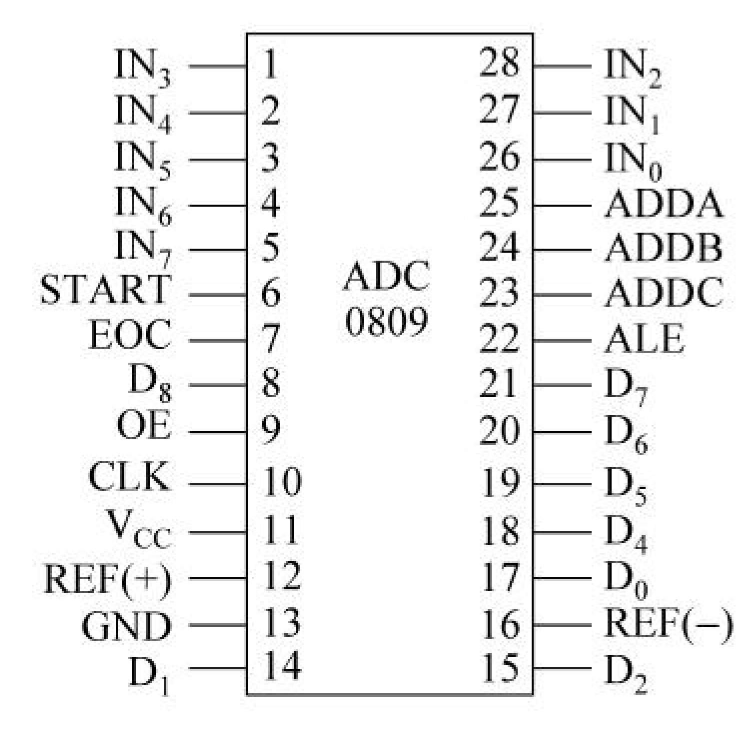


图 8-16 ADC0809 外部引线图

(7) OE：读允许信号，高电平有效。在其有效期间，CPU 将转换后的数字量读入。

(8) CLK：时钟输入端。

(8) REF(＋)，REF(－)：参考电压输入端。

(10) V_{CC}：5V 电源输入。

(11) GND：地线。

ADC0809 需要外接参考电源和时钟。外接时钟频率为 10kHz～1.2MHz。

2) ADC0809 的内部结构

ADC0809 的内部结构框图如图 8-17 所示，它主要由 3 部分组成。

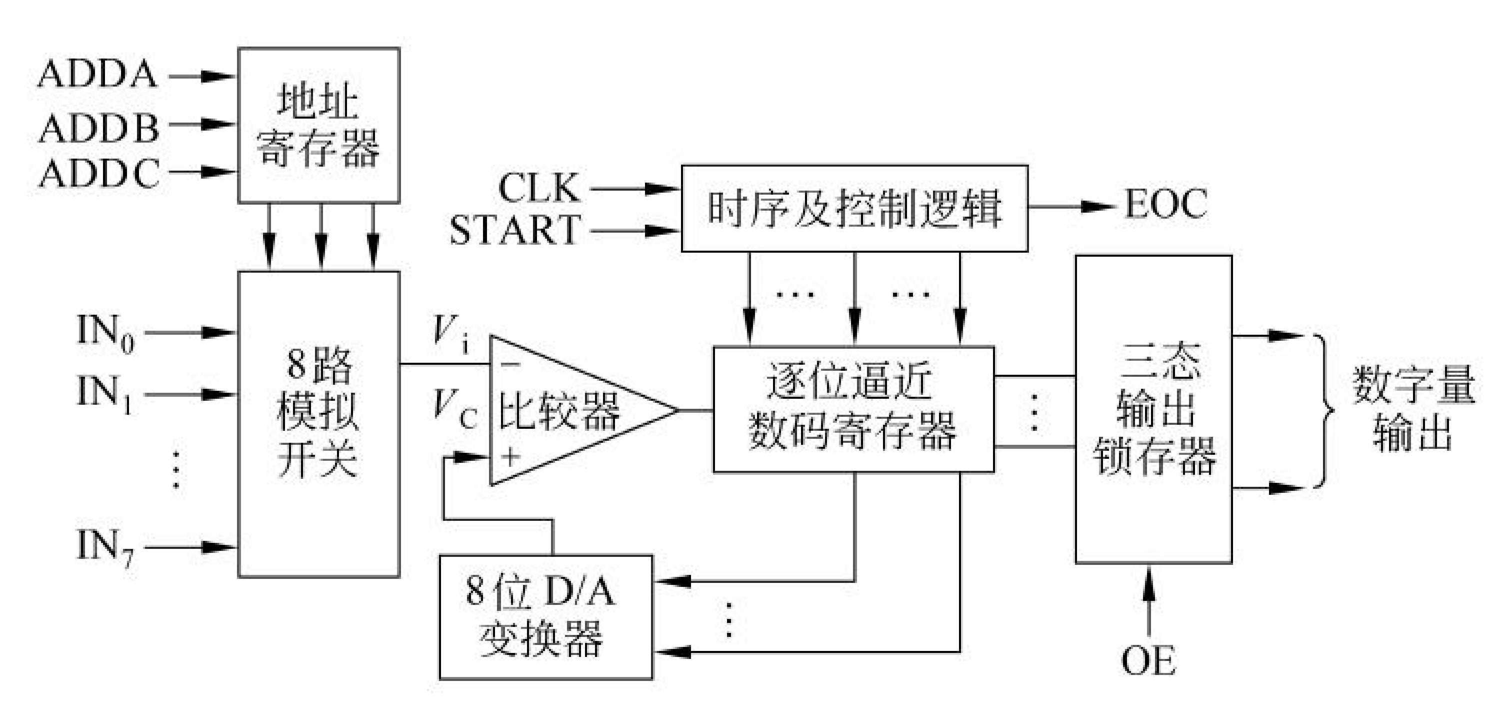


图 8-17　ADC0809 内部结构框图

(1) 模拟输入选择部分——包括一个 8 路模拟开关和地址锁存与译码电路。输入的 3 位通道地址信号由锁存器锁存，经译码电路译码后控制模拟开关选择相应的模拟输入。地址编码与输入通道的关系如表 8-1 所示。

表 8-1　输入通道和地址

对应模拟通道	ADDC	ADDB	ADDA	对应模拟通道	ADDC	ADDB	ADDA
IN_0	0	0	0	IN_4	1	0	0
IN_1	0	0	1	IN_5	1	0	1
IN_2	0	1	0	IN_6	1	1	0
IN_3	0	1	1	IN_7	1	1	1

(2) 转换器部分——主要包括比较器、8 位 D/A 转换器、逐位逼近寄存器以及控制逻辑电路等。

(3) 输出部分——包括一个 8 位三态输出缓冲器。

2. ADC0809 的工作过程

ADC0809 的工作时序如图 8-18 所示。外部时钟信号通过 CLK 端进入其内部控制逻辑电路，作为转换时的时间基准。由时序图可以看出 ADC0809 的工作过程如下。

(1) 首先 CPU 发出 3 位通道地址信号 ADDC、ADDB、ADDA。

(2) 在通道地址信号有效期间，使 ALE 引脚上产生一个由低到高的电平变化，即脉冲上跳沿，它将输入的 3 位通道地址锁存到内部地址锁存器。

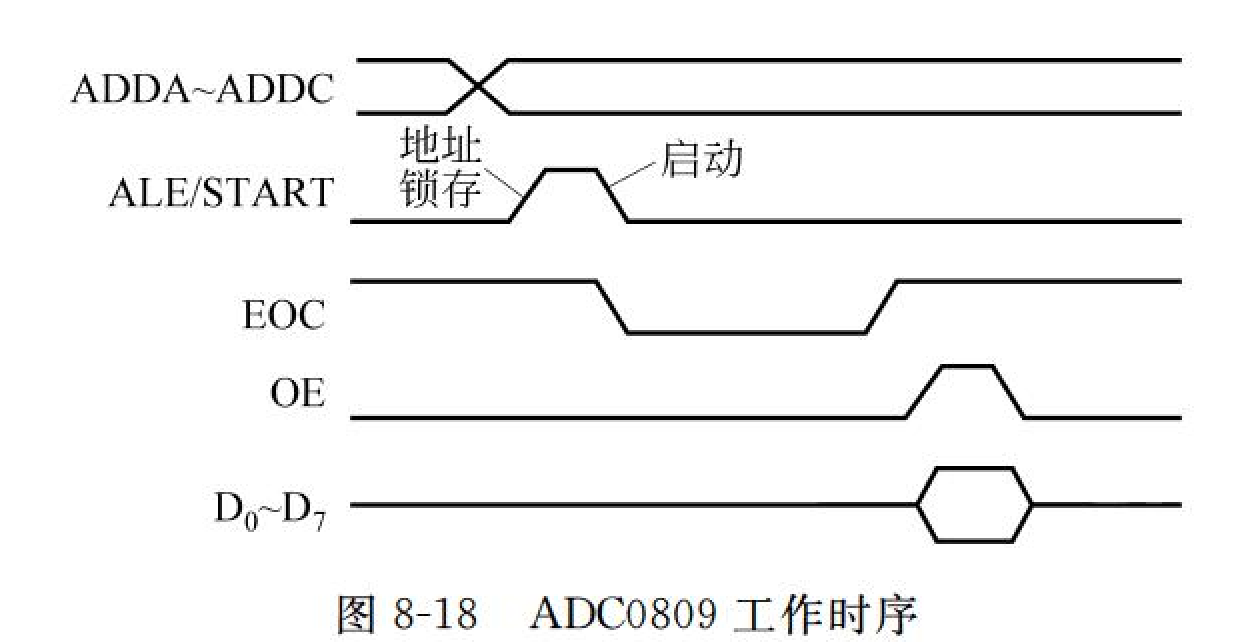


图 8-18 ADC0809 工作时序

(3) 接着给 START 引脚加上一个由高到低变化的电平,启动 A/D 变换。

(4) 变换开始后,EOC 引脚呈现低电平,一旦变换结束,EOC 又重新变为高电平。

(5) CPU 在检测到 EOC 变高后,输出一个正脉冲到 OE 端,将转换结果取走。

3. ADC0809 的主要技术指标

ADC0809 的主要技术指标如下。

(1) 分辨率:8 位。

(2) 转换时间:100μs。

(3) 电源:单电源 0~+5V。

4. ADC0809 与系统的连接方法

1) 输入模拟量

输入模拟信号分别连接到 IN_0~IN_7 端。当前要转换哪一路通过 ADDC~ADDA 的不同编码来选择。ADC0809 内部包括有地址锁存器,CPU 可通过一个输出接口(如 74LS273、74LS373、8255 等)把通道地址编码送到通道地址信号端。

2) 数据信号

由图 8-17 可知,ADC0809 芯片的 D_7~D_0 输出端带有三态缓冲器,所以它可以直接连接到系统数据总线上。但考虑到驱动及隔离的因素,通常总是用一个输入接口与系统连接。

3) 启动变换信号

ADC0809 采用脉冲启动方式启动变换信号。通常将 START 和 ALE 连接在一起作为一个端口看待。因为 ALE 是上升沿有效,而 START 是下降沿有效,这样连接就可用一个正脉冲来完成通道地址锁存和启动转换两项工作。初始状态下使该端口为低电平。当通道地址信号输出后,CPU 往该端口送出一个正脉冲,其上升沿锁存地址,下降沿启动变换。

4) 状态信号 EOC 的连接

判断一次 A/D 转换是否结束有以下几种方式。

(1) 软件延时方式。编写延时程序,使延时时间≥A/D 变换时间,延时时间到,读取转换结果。一般来说,这种方式的实时性要差一些。

(2) 查询方式。转换过程中,CPU 通过程序不断地读取 EOC 端的状态,在读到其状态为"1"时,则表示一次转换结束。

(3) 中断控制方式。可将 ADC0809 的 EOC 端接到中断控制器 8259A 的中断请求

输入端,当 EOC 端由低电平变为高电平时(转换结束),即产生中断请求。CPU 在收到该中断请求信号后,读取转换结果。由于 A/D 变换的过程需要一定的时间,所以采用中断控制方式 CPU 效率最高。

ADC0809 与系统的连接图如图 8-19 所示。为尽量使 ADC0809 少占用地址资源,将其各控制信号和输出端都通过输入输出接口与系统相连。图中用 74LS244 作为输入端口,74LS273 作为输出端口。若采用第 7 章介绍的可编程并行接口 8255 芯片,则其电路连接如图 8-20 所示。图中使 8255 工作在方式 0 下,A 口作为转换结果的输入口,B 口和 C 口连接各控制信号。

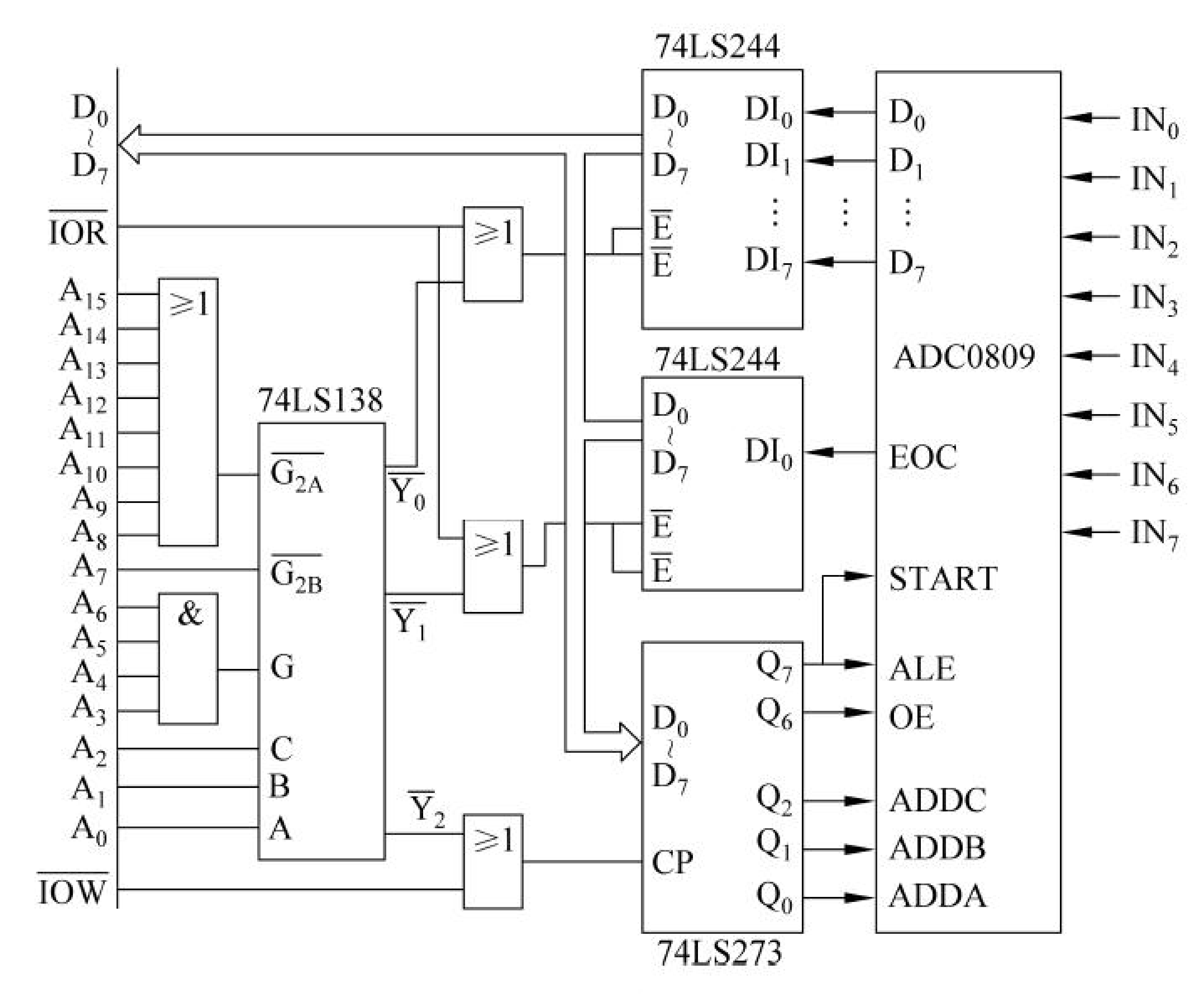


图 8-19　ADC0809 与系统连接图 1

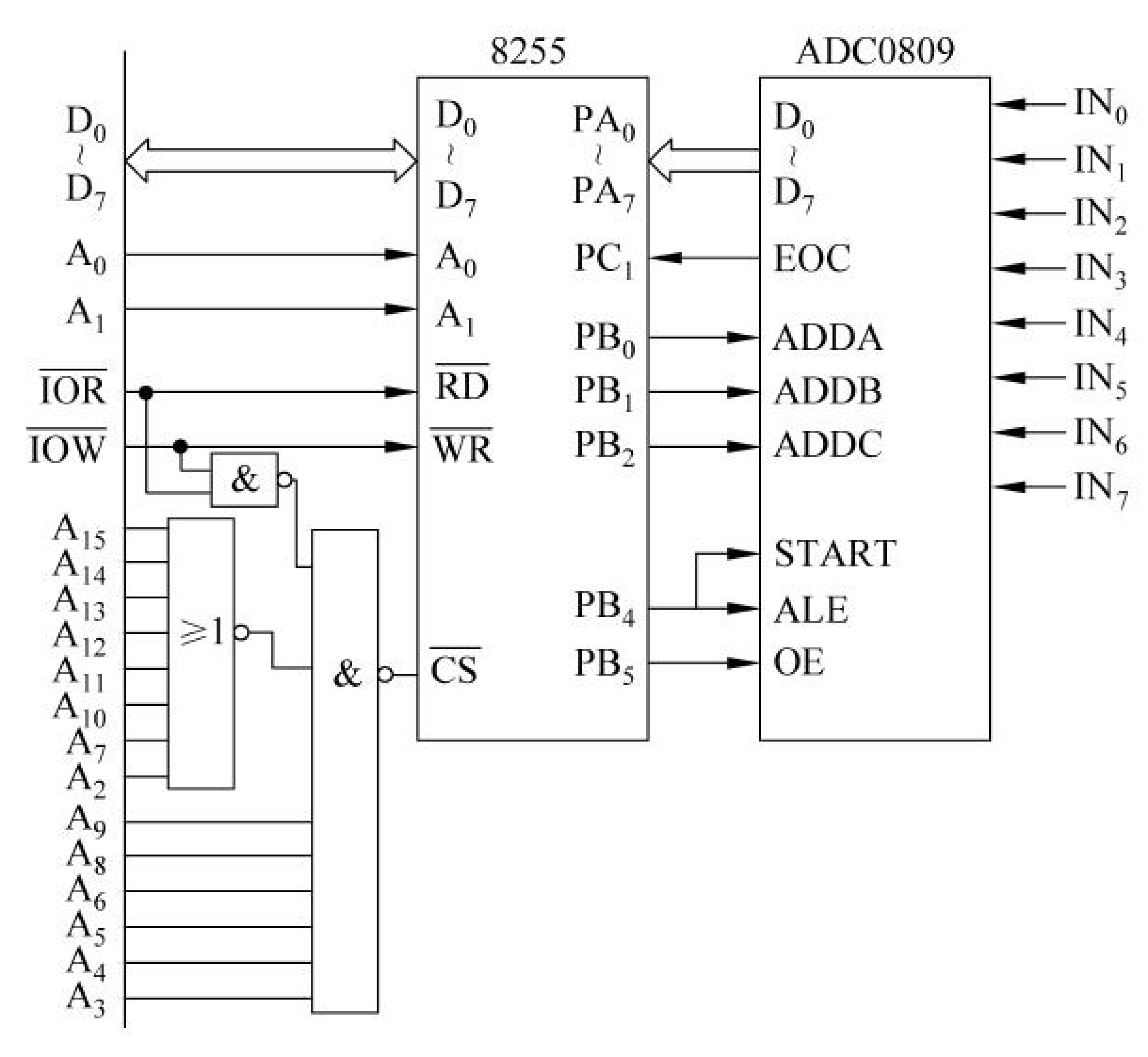


图 8-20　ADC0809 与系统连接图 2

注意：若使用 8255 作为输入输出接口，必须首先给 8255 初始化。

5. ADC0809 的应用

ADC0809 主要用于数据采集系统中，可以实现对 8 路模拟输入信号的循环数据采集。

【例 8-4】 以图 8-20 为例，编写 8 路模拟量的循环数据采集程序，并将转换结果（数字量）放在 DATA 为首的内存单元中。

题目分析：

由图 8-20 可知，8255 的地址为 0378H～037BH。A、B、C 这 3 个端口均工作在方式 0，A 口作为输入口，输入转换后的结果；B 口作为输出口，用来输出通道地址、发出地址锁存信号和启动转换信号；C 口低 4 位为输入口，用来读取转换状态，高 4 位没有使用。

设计程序如下：

```
INIT_8255  PROC  NEAR                        ;8255初始化
           MOV   DX, 037BH
           MOV   AL, 91H                     ;A、B、C均为方式 0,A入、B出、C入
           OUT   DX, AL
           RET
INIT_8255  ENDP
```

数据采集：

```
START: MOV   AX, SEG DATA
       MOV   DS, AX
       MOV   SI, OFFSET DATA
       CALL  INIT_8255;                      ;初始化 8255
       MOV   BL, 0                           ;通道号,初始指向第 0路
       MOV   CX, 8                           ;共采集 8次,每路采集 1次
AGAIN: MOV   AL, BL
       MOV   DX, 0379H
       OUT   DX, AL                          ;送通道地址
       OR    AL, 10H
       OUT   DX, AL                          ;送 ALE信号 (上升沿)
       AND   AL, 0EFH
       OUT   DX, AL                          ;输出 START信号 (下降沿)
       NOP                                   ;空操作等待转换
       MOV   DX, 037AH
WAIT1: IN    AL, DX                          ;读 EOC状态
       AND   AL, 02H
       JZ    WAIT1                           ;若 EOC为低电平则等待
       MOV   DX, 0379H
       MOV   AL, BL
```

```
OR    AL, 20H
OUT   DX, AL                 ;EOC端为高电平则输出读允许信号 OE=1
MOV   DX, 0378H
IN    AL, DX                 ;读入变换结果
MOV   [SI], AL               ;将转换的数字量送存储器
INC   SI                     ;修改指针
INC   BL                     ;修改通道地址值
LOOP  AGAIN                  ;若未采集完则再采集下一路数据
MOV   DX, 0379H
MOV   AL, 0
OUT   DX, AL                 ;若 8路数据已采集完则回到初始状态
HLT
```

以上就是 8 路模拟量的数据采集程序，每执行一次该程序，数据段中以 DATA 为首地址的顺序单元中就会存放 IN_0～IN_7 端模拟信号所对应的 8 位数字量。该程序通过查询 EOC 端口的状态来判断是否一次变换结束。用中断或延时的方法来决定是否转换结束的程序留作读者自行考虑。

另外，在上述程序中，是利用程序对读允许信号 OE 进行控制的。实际上，由于借用了数字 I/O 接口，也可将该端直接接到＋5V 电源上，这样就可以将程序中对 OE 控制的指令删去。

以上，通过典型的 A/D 转换器芯片 ADC0809，介绍了 A/D 转换器的工作原理、与系统的连接及其应用等，希望读者能够熟练地掌握它的使用方法，并由此在碰到类似芯片时也能较容易地熟悉它们。

学习了数字并行接口，又学习了模拟并行接口，现在，我们可以设计基于模拟监测传感器的“家庭安全防盗系统”了。需要说明的是，一个完整的安全防盗系统会涉及部分非本书所涵盖的知识(如传感器原理、信号处理、各类执行机构工作原理、控制技术等)。因此，以下的硬件线路设计依然属于“原理示意图”，参数为假设值，软件设计则仅涵盖核心控制程序。

“家庭安全防盗系统”设计方案示例 4：

设计基于如下假设：监测装置采用红外式传感器。当有人体进入监测区域时，假设传感器输出 3～5V 的模拟电压信号。

基本方案：选择 ADC0809 作为模拟接口，将红外传感器输出的模拟信号转换为数字信号，再通过 8255 接口输入到系统。在“方案示例 3”的基础上，设计如图 8-21 所示的“家庭安全防盗系统”。

安全防盗系统的主要功能是：当需要时(例如人员外出)，将开关 K 闭合，启动布防。之后，系统开始依次循环采集各监测传感器值。若传感器输出电压值在 3～5V(对应数字量为 153～255)，则启动报警输出。即在 8253 定时/计算器的 OUT0 端输出频率为 1Hz 的连续方波信号，使报警器发声；在 OUT1 端输出 2Hz 方波信号，控制报警灯闪烁。报警程序基本控制流程如图 8-22 所示。

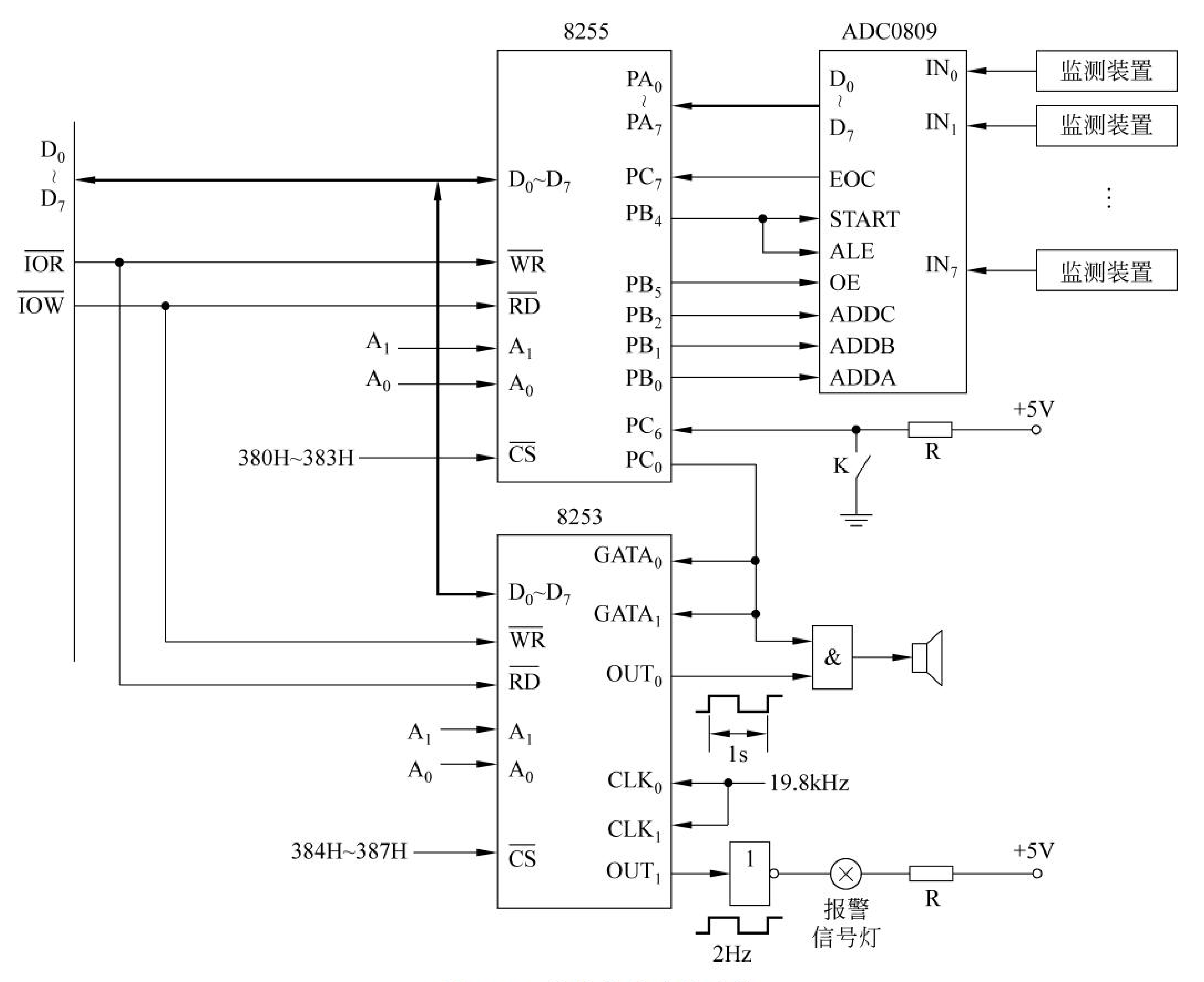


图 8-21　家庭安全防盗系统

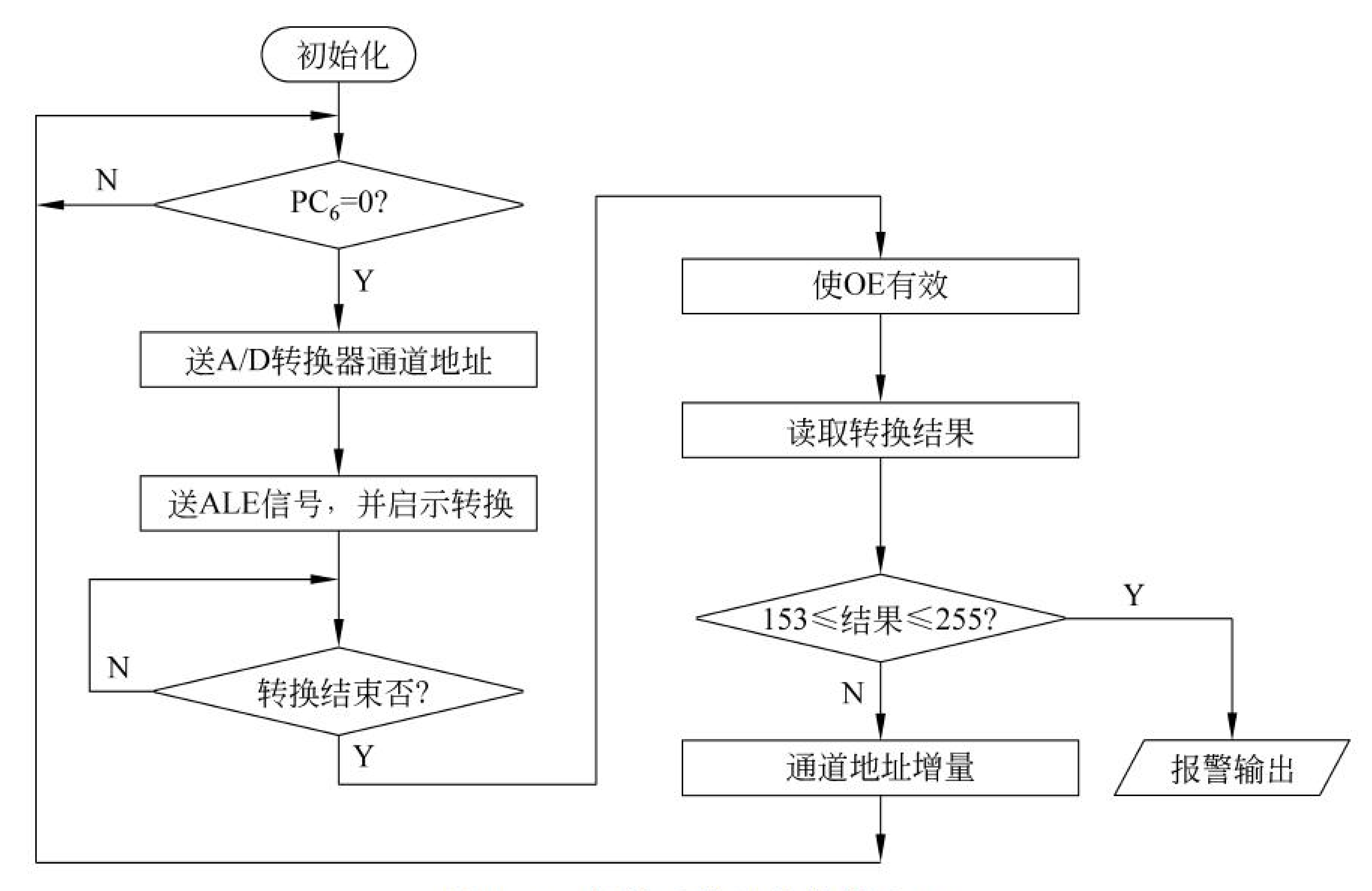


图 8-22　报警系统基本控制流程

有了图 8-21 和硬件线路图和图 8-22 所示的控制流程，现在就可以综合本书所介绍的知识，完成系统的软、硬件设计了。

习　题

8.1　试说明将一个工业现场的非电物理量转换为计算机能够识别的数字信号主要需经过哪几个过程。

8.2　什么是 A/D 转换器？什么是 D/A 转换器？它们的主要作用是什么？

8.3　D/A 转换器主要有哪些技术指标？影响其转换误差的主要因素是什么？

8.4　对于一个 10 位的 D/A 转换器，其分辨率是多少？如果输出满刻度电压值为 5V，那么一个最低有效位对应的电压值等于多少？

8.5　某一测控系统要求计算机输出模拟控制信号的分辨率必须达到 1‰，则应选用的 D/A 芯片的位数至少是多少？

8.6　DAC0832 在逻辑上由哪几个部分组成？可以工作在哪几种模式下？不同工作模式在线路连接上有什么区别？

8.7　如果要求同时输出 3 路模拟量，则 3 片同时工作的 DAC0832 最好采用哪一种工作模式？

8.8　某 8 位 D/A 转换器，输出电压为 0～5V。当输入的数字量为 40H、80H 时，其对应的输出电压分别是多少？

8.9　ADC0809 是完成什么功能的芯片？试说明它的变换原理。

8.10　设 DAC0832 工作在单缓冲模式下，端口地址为 034BH，输出接运算放大器。试画出其与 8088 系统的线路连接图，并编写输出三角波的程序段。

8.11　对 8 位、10 位和 12 位的 A/D 转换器，当满刻度输入电压为 5V 时，其量化间隔各为多少？绝对量化误差又为多少？

8.12　某工业现场的 3 个不同点的压力信号经压力传感器、变送器及信号处理环节等分别送入 ADC0809 的 IN_0、IN_1 和 IN_2 端。计算机巡回检测这 3 点的压力并进行控制。试编写数据采集程序。

8.13　设被测温度的变化范围为 0～100℃，若要求测量误差不超过 0.1℃，应选用分辨率为多少位的 A/D 转换器？

8.14　某 11 位 A/D 转换器的引线及工作时序如图 8-23 所示，利用不小于 1μs 的后沿脉冲(START)启动变换。当 $\overline{BUSY}$ 端输出低电平时表示正在变换，$\overline{BUSY}$ 变高则变换结束。为获得变换好的二进制数据，必须使 $\overline{OE}$ 为低电平。现将该 A/D 转换器与 8255 相连，8255 的地址范围为 03F4H～03F7H。试画线路连接图，编写包括 8255 初始化程序在内的、完成一次数据变换并将数据存放在 DATA 中的程序。

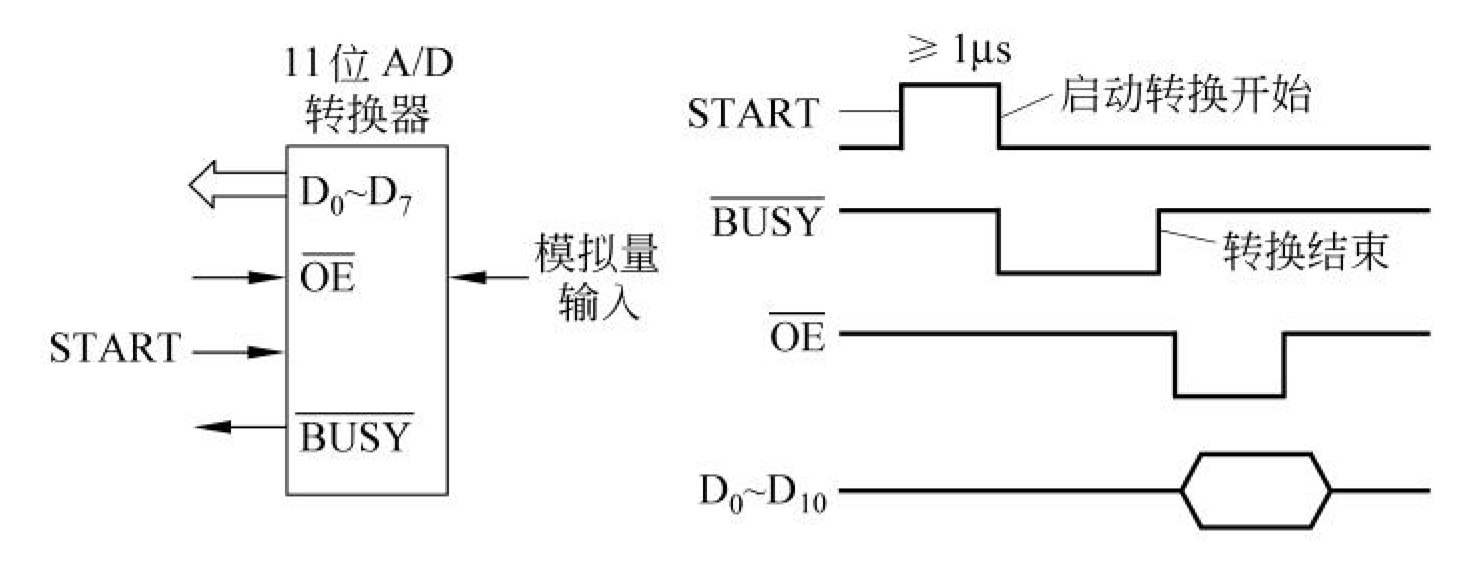


图 8-23　11 位 A/D 转换器的引线及工作时序

附录 A　ASCII 码表及其中控制符号的定义

A.1　ASCII 码表

行	列	0	1	2	3	4	5	6	7
	高位 低位	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	、	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	"	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	'	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	—	=	M	]	m	}
E	1110	SO	RS	.	>	N	Ω	n	～
F	1111	SI	US	/	?	O	_	o	DEL

注：表中的 00H～1FH 以及 7FH 为控制符，不可显示；其余的为可显示字符。

A.2　ASCII 码表中控制符号的定义

NUL	Null	空白	DLE	Data Link Escape	转义
SOH	Start Of Heading	标题开始	DC1	Device Control 1	设备控制 1
STX	Start Of Text	正文开始	DC2	Device Control 2	设备控制 2
ETX	End Of Text	正文结束	DC3	Device Control 3	设备控制 3
EOT	End Of Transmit	传输结束	DC4	Device Control 4	设备控制 4
ENQ	Enquiry	询问	NAK	Negative Acknowledge	否定
ACK	Acknowledge	承认	SYN	Synchronize	同步
BEL	Bell	响铃	ETB	End of Transmitted Block	信息组结束

续表

BS	Back Space	退格	CAN	Cancel	作废
HT	Horizontal Tab	横向制表	EM	End of Medium	纸尽
LF	Line Feed	换行	SUB	Substitute	取代
VT	Vertical Tab	纵向制表	ESC	Escape	换码
FF	Form Feed	换页	FS	File Separator	文件分隔符
CR	Carriage Return	回车	GS	Group Separator	组分隔符
SO	Shift Out	移出	RS	Record Separator	记录分隔符
SI	Shift In	移入	US	Unit Separator	单元分隔符
SP	Space	空格	DEL	Delete	删除

附录 B　8088 CPU 部分引脚信号功能

B.1　$\overline{SS_0}$、IO/$\overline{M}$、DT/$\overline{R}$ 的组合及对应的操作

IO/$\overline{M}$	DT/$\overline{R}$	$\overline{SS_0}$	操　　作	IO/$\overline{M}$	DT/$\overline{R}$	$\overline{SS_0}$	操　　作
1	0	0	发中断响应信号	0	0	0	取指令
1	0	1	读 I/O 端口	0	0	1	读内存
1	1	0	写 I/O 端口	0	1	0	写内存
1	1	1	暂停	0	1	1	无作用

B.2　$\overline{S_2}$、$\overline{S_1}$、$\overline{S_0}$的组合及对应的操作

$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	操　　作	$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	操　　作
0	0	0	发中断响应信号	1	0	0	取指令
0	0	1	读 I/O 端口	1	0	1	读存储器
0	1	0	写 I/O 端口	1	1	0	写存储器
0	1	1	暂停	1	1	1	无作用

B.3　QS_1、QS_0 的组合及对应的操作

QS_1	QS_0	操　　作	QS_1	QS_0	操　　作
0	0	无操作	1	0	队列空
0	1	队列中操作码的第一个字节	1	1	队列中非第一个操作码字节

附录C　8086/8088 指令执行时间及指令简表

C.1　常用指令执行时间

指　　令		所需时钟周期数	访问内存次数
MOV	累加器到内存	10(14)	1
	内存到累加器	10(14)	1
	寄存器到寄存器	2	0
	内存到寄存器	8(12)＋EA	1
	寄存器到内存	9(13)＋EA	1
MOV	立即数到寄存器	4	0
	立即数到内存	10(14)＋EA	1
	寄存器到段寄存器	2	0
	内存到段寄存器	8(12)＋EA	1
	段寄存器到寄存器	2	0
	段寄存器到内存	9(13)＋EA	1
ADD 或 SUB	寄存器到寄存器	3	0
	内存到寄存器	9(13)＋EA	1
	寄存器到内存	16(24)＋EA	2
	立即数到寄存器	4	0
	立即数到内存	17(25)＋EA	2
MUL	累加器乘 8 位寄存器	70～77	0
	累加器乘 16 位寄存器	118～133	0
	累加器和内存字节乘	(76～83)＋EA	1
	累加器和内存字乘	[124(128)～139(143)]＋EA	1
IMUL	累加器乘 8 位寄存器	80～98	0
	累加器乘 16 位寄存器	128～154	0
	累加器和内存字节乘	(86～104)＋EA	1
	累加器和内存字乘	[134(138)～160(164)]＋EA	1
DIV	除数在 8 位寄存器中	80～90	0
	除数在 16 位寄存器中	144～162	0
	除数为 8 位内存数	(86～96)＋EA	1
	除数为 16 位内存数	[150(154)～168(172)]＋EA	1
IDIV	除数在 8 位寄存器中	101～112	0
	除数在 16 位寄存器中	165～184	0
	除数为 8 位内存数	(107～118)＋EA	1
	除数为 16 位内存数	[171(175)～190(194)]＋EA	1

续表

指　　令		所需时钟周期数	访问内存次数
循环和移位	在寄存器中移1位 在寄存器中移若干位 内存数据移1位 内存数据移若干位	2 8+4*位数 15(23)+EA 20(28)+EA+4*位数	
JMP	段内/段间直接转移 段内间接转移 段间间接转移	15 8(12)+EA 24(32)+EA	
条件转移	JCXZ 其他条件转移指令	6(不转移) 18(转移) 4(不转移) 16(转移)	

注：

(1) 表中EA表示偏移地址，小括号内的数为8088进行字操作的时钟数，因为8088的数据线只有8位，每个总线周期只能传送一个字节，所以对字操作要再加上4个时钟周期；

(2) 对条件转移指令，若条件满足，执行的时间比较长，因为要产生转移，就要包括取下一条指令所需的时间，若条件不满足，执行时间就较短，因为此时不产生转移，而是执行下一条指令。

计算偏移地址EA所需时间

寻 址 方 式		计算EA所需时钟数
直接寻址		6
寄存器间接寻址		5
寄存器相对寻址		9
基址、变址寻址	[BX+SI]、[BX+DI]	7
	[BP+SI]、[BP+DI]	8
基址、变址加相对寻址	[BX+SI+位移量]、[BP+DI+位移量]	11
	[BX+DI+位移量]、[BP+SI+位移量]	12

注：

(1) 若有段超越，则需再加上两个时钟周期；

(2) 寻址方式的介绍参见3.2节。

C.2　8086/8088指令简表

汇 编 格 式	指令的操作
1. 数据传送指令	
MOV　*dest*,*source*	数据传送
CBW	字节转换成字
CWD	字转换成双字
LAHF	FLAGS低8位装入AH寄存器
SAHF	AH寄存器内容送到FLAGS低8位
LDS　*dest*,*source*	设定数据段指针

续表

汇 编 格 式	指令的操作
1. 数据传送指令	
LES *dest*,*source*	设定附加段指针
LEA *dest*,*source*	装入有效地址
PUSH *source*	将一个字压入栈顶
POP *dest*	将一个字从栈顶弹出
PUSHF	将标志寄存器 FLAGS 的内容压入栈顶
POPF	将栈顶内容弹出到标志寄存器 FLAGS
XCHG *dest*,*source*	交换
XLAT *source*	表转换
2. 算术运算指令	
AAA	加法的 ASCII 调整
AAD	除法的 ASCII 调整
AAM	乘法的 ASCII 调整
AAS	减法的 ASCII 调整
DAA	加法的十进制调整
DAS	减法的十进制调整
MUL *source*	无符号乘法
IMUL *source*	整数乘法
DIV *source*	无符号除法
IDIV *source*	整数除法
ADD *dest*,*source*	加法
ADC *dest*,*source*	带进位加
SUB *dest*,*source*	减法
SBB *dest*,*source*	带借位减
CMP *dest*,*source*	比较
INC *dest*	加 1
DEC *dest*	减 1
NEG *dest*	求补
3. 逻辑运算指令	
AND *dest*,*source*	逻辑“与”
OR *dest*,*source*	逻辑“或”
XOR *dest*,*source*	逻辑“异或”
NOT *dest*	逻辑“非”
TEST *dest*,*source*	测试(非破坏性逻辑“与”)
4. 移位指令	
RCL *dest*,*count*	通过进位循环左移
RCR *dest*,*count*	通过进位循环右移
ROL *dest*,*count*	循环左移
ROR *dest*,*count*	循环右移
SHL/SAL *dest*,*count*	逻辑左移/算术左移
SHR *dest*,*count*	逻辑右移
SAR *dest*,*count*	算术右移

续表

汇 编 格 式	指令的操作
5. 串操作指令	
MOVS/MOVSB/MOVSW *dest*,*source*	字符串传送
CMPS/CMPSB/CMPSW *dest*,*source*	字符串比较
LODS/LODSB/LODSW *source*	装入字节串或字串到累加器
STOS/STOSB/STOSW *dest*	存储字节串或字串
SCAS/SCASB/SCASW *dest*	字符串扫描
6. 程序控制指令	
CALL *dest*	调用一个过程(子程序)
RET [弹出字节数(必须为偶数)]	从过程(子程序)返回
INT *int_type*	软件中断
INTO	溢出中断
IRET	从中断返回
JMP *dest*	无条件转移
JG/JNLE *short_label*	大于或不小于等于转移
JGE/JNL *short_label*	大于等于或不小于转移
JL/JNGE *short_label*	小于或不大于等于转移
JLE/JNG *short_label*	小于等于或不大于转移
JA/JNBE *short_label*	高于或不低于等于转移
JAE/JNB *short_label*	高于等于或不低于转移
JB/JNAE *short_label*	低于或不高于等于转移
JBE/JNA *short_label*	低于等于或不高于转移
JO *short_label*	溢出标志为 1 转移(溢出转移)
JNO *short_label*	溢出标志为 0 转移(无溢出转移)
JS *short_label*	符号标志为 1 转移(结果为负转移)
JNS *short_label*	符号标志为 0 转移(结果为正转移)
JC *short_label*	进位标志为 1 转移(有进位转移)
JNC *short_label*	进位标志为 0 转移(无进位转移)
JZ/JE *short_label*	零标志为 1 转移(等于或为 0 转移)
JNZ/JNE *short_label*	零标志为 0 转移(不等于或不为 0 转移)
JP/JPE *short_label*	奇偶标志为 1 转移(结果中有偶数个 1 转移)
JNP/JPO *short_label*	奇偶标志为 0 转移(结果中有奇数个 1 转移)
JCXZ *short_label*	若 CX=0 则转移
LOOP *short_label*	CX≠0 时循环
LOOPE/LOOPZ *short_label*	CX≠0 且 ZF=1 时循环
LOOPNE/LOOPNZ *short_label*	CX≠0 且 ZF=0 时循环
STC	进位标志置 1
CLC	进位标志置 0
CMC	进位标志取反
STD	方向标志置 1
CLD	方向标志置 0

续表

汇 编 格 式	指令的操作
6. 程序控制指令	
STI	中断标志置 1(允许可屏蔽中断)
CLI	中断标志置 0(禁止可屏蔽中断)
ESC	CPU 交权
HLT	停机
LOCK	总线封锁
NOP	无操作
WAIT	等待至 $\overline{\text{TEST}}$ 信号有效为止
7. 输入输出指令	
IN *acc*,*source*	从外设接口输入字节或字
OUT *dest*,*acc*	向外设接口输出字节或字

注:

dest:目的操作数、目的串;

source:源操作数、源串;

acc:累加器;

count:计数值;

int_type:中断类型号;

short_label:短距离标号。

附录 D　8086/8088 微机的中断

D.1　中断类型分配

类　　别	中断类型码(Hex)	功　　能
软件自陷和 NMI 中断	0	除法错
	1	单步
	2	NMI 中断
	3	断点
	4	溢出
	5	屏幕复制
	6、7	未使用
主 8259 管理的中断(可屏蔽中断)	8	系统定时器
	9	键盘
	A	未使用(从 8259A 与此中断级联)
	B	COM2
	C	COM1
	D	并口 2(打印机)
	E	软盘驱动器
	F	并口 1(打印机)
ROM-BIOS 软中断	10	屏幕显示
	11	检测系统配置
	12	检测存储器容量
	13	磁盘 I/O
	14	异步通信 I/O
	15	盒式磁带机,I/O 系统扩展
	16	键盘 I/O
	17	打印机 I/O
	18	ROM-BASIC 入口
	19	系统自举(冷启动)
	1A	日时钟 I/O
供用户链接的中断	1B	键盘 Ctrl＋Break 中断
	1C	定时器产生的中断(每 55ms 产生一次)
数据表指针	1D	显示器初始化参数
	1E	软盘参数
	1F	显示图形字符
DOS 软中断	20	程序正常结束
	21	系统功能调用
	22	程序结束退出
	23	Ctrl＋Break 退出

续表

类　　别	中断类型码(Hex)	功　　能
DOS 软中断	24 25 26 27 28～2E 2F 30～3F	严重错误处理 绝对磁盘读功能 绝对磁盘写功能 程序驻留并退出 DOS 保留 假脱机打印 DOS 保留
杂类	40 41 42～5F 60～6F	软盘 I/O 重定向 硬盘参数 系统保留 保留给用户使用
从 8259 管理的中断(可屏蔽中断)	70 71 72 73 74 75 76 77	实时时钟 IRQ_9(INT 0AH 重定向) IRQ_{10}(保留) IRQ_{11}(保留) IRQ_{12}(保留) 协处理器 硬盘控制器 IRQ_{15}(保留)
其他	78～7F 80～F0 F1～FF	未使用 BASIC 占用 未使用

D.2　DOS 软中断

中　断	功　　能	入 口 参 数	出 口 参 数
INT 20H	程序正常退出		
INT 21H	系统功能调用	AH=功能号 其他参数随功能而异(见 C.3)	随功能而异(见 C.3)
INT 22H	程序结束		
INT 23H	Ctrl-Break 退出		
INT 24H	严重错误处理		
INT 25H	绝对磁盘读	AL=盘号 CX=读的扇区数 DX=起始逻辑扇区号 DS:BX=缓冲区首址	CF=1 出错
INT 26H	绝对磁盘写	AL=盘号 CX=写的扇区数 DX=起始逻辑扇区号 DS:BX=缓冲区首址	CF=1 出错
INT 27H	驻留退出		

D.3 DOS系统功能调用简表

功能号	功　　能	入口参数	出口参数
1. 设备管理功能			
01H	键盘输入		AL＝输入字符
02H	显示器输出	DL＝输出字符	
03H	串行设备输入字符		AL＝输入字符
04H	串行设备输出字符	DL＝输出字符	
05H	打印机输出	DL＝输出字符	
06H	直接控制台I/O	DL＝FFH(输入) DL＝输出字符(输出)	AL＝输入字符
07H	直接控制台输入(无回显)		AL＝输入字符
08H	键盘输入(无回显)		AL＝输入字符
09H	显示字符串	DS:DX＝字符缓冲区首址	
0AH	带缓冲的键盘输入(字符串)	DS:DX＝键盘缓冲区首址	
0BH	检查标准输入状态		AL＝0　无键入 AL＝FFH 有键入
0CH	清除键盘缓冲区,然后输入	AL＝功能号(1、6、7、8、A)	(与指定的功能相同)
0DH	刷新DOS磁盘缓冲区		
0EH	选择磁盘	DL＝盘号	AL＝系统中盘的数目
19H	取当前盘盘号		AL＝盘号
1AH	设置磁盘传送缓冲区(DTA)	DS:DX＝DTA首址	
1BH	取当前盘文件分配表(FAT)信息		DS:BX＝盘类型字节地址 DX＝FAT表项数 AL＝每簇扇区数 CX＝每扇区字节数
1CH	取指定盘文件分配表(FAT)信息	DL＝盘号	(同上)
2EH	置写校验状态	DL＝0, AL＝状态(0关,1开)	AL＝0成功, AL＝FFH失败
54H	取写校验状态		AL＝状态(0关,1开)
36H	取盘剩余空间	DL＝盘号	BX＝可用簇数 DX＝总簇数 AX＝每簇扇区数 CX＝每扇区字节数
2FH	取磁盘传送缓冲区(DTA)首址		ES:BX＝DTA首址

功能号	功　　能	入口参数	出口参数
2. 文件管理功能			
29H	建立文件控制块 FCB	DS:SI=文件名字符串首址 ES:DI=FCB 首址 AL=0EH 非法字符检查	ES:DI=格式化后的 FCB 首址 AL=0 标准文件 AL=1 多义文件 AL=FFH 非法盘符
16H	建立文件(FCB 方式)	DS:DX=FCB 首址	AL=0 成功 AL=FFH 目录区满
0FH	打开文件(FCB 方式)	DS:DX=FCB 首址	AL=0 成功 AL=FFH 未找到
10H	关闭文件(FCB 方式)	DS:DX=FCB 首址	AL=0 成功 AL=FFH 已换盘
13H	删除文件(FCB 方式)	DS:DX=FCB 首址	AL=0 成功 AL=FFH 未找到
14H	顺序读一个记录	DS:DX=FCB 首址	AL=0 成功 AL=1 文件结束 AL=3 缓冲不满
15H	顺序写一个记录	DS:DX=FCB 首址	AL=0 成功 AL=FFH 盘满
21H	随机读一个记录	DS:DX=FCB 首址	AL=0 成功 AL=1 文件结束 AL=3 缓冲不满
22H	随机写一个记录	DS:DX=FCB 首址	AL=0 成功 AL=FFH 盘满
27H	随机读多个记录	DS:DX=FCB 首址 CX=记录数	AL=0 成功 AL=1 文件结束 AL=3 缓冲不满
28H	随机写多个记录	DS:DX=FCB 首址 CX=记录数	AL=0 成功 AL=FFH 盘满
24H	置随机记录号	DS:DX=FCB 首址	
3CH	建立文件(文件号方式)	DS:DX=文件号首址 CX=文件属性	若 CF=0,AX=文件号 否则失败,AX=错误代码
3DH	打开文件(文件号方式)	DS:DX=文件号首址 AL=0 只读 AL=1 只写 AL=2 读/写	若 CF=0,AX=文件号 否则失败, AX=错误代码
3EH	关闭文件(文件号方式)	BX=文件号	CF=0 成功, 否则失败
41H	删除文件(文件号方式)	DS:DX=文件号首址	若 CF=0,成功 否则失败,AX=错误代码

续表

功能号	功　　能	入口参数	出口参数
2. 文件管理功能			
3FH	读文件(文件号方式)	BX=文件号 CX=读的字节数 DS:DX=缓冲区首址	AX=实际读的字节数
40H	写文件(文件号方式)	BX=文件号 CX=写的字节数 DS:DX=缓冲区首址	AX=实际写的字节数
42H	移动文件读写指针	BX=文件号 CX:DX=位移量 AL=0 从文件头开始移动 AL=1 从当前位置移动 AL=2 从文件尾倒移	若 CF=0,成功 DX:AX=新的指针位置 否则失败 AX=1 无效的移动方法 AX=6 无效的文件号
45H	复制文件号	BX=文件号 1	若 CF=0,AX=文件号 2 否则失败,AX=错误代码
46H	强制复制文件号	BX=文件号 1 CX=文件号 2	若 CF=0,CX=文件号 1 否则失败,AX=错误代码
4BH	装入一个程序	DS:DX=程序路径名首址 ES:BX=参数区首址 AL=0 装入后执行 AL=3 仅装入	若 CF=0,成功 否则失败
44H	设备文件 I/O 控制	BX=文件号 AL=0 取状态 AL=1 置状态 DX AL=2 读数据 * AL=3 写数据 * AL=6 取输入状态 AL=7 取输出状态 (* DS:DX=缓冲区首址,CX=读写的字节数)	DX=状态
3. 目录操作功能			
11H	查找第一个匹配文件(FCB 方式)	DS:DX=FCB 首址	AL=0 成功 AL=FFH 未找到
12H	查找下一个匹配文件(FCB 方式)	DS:DX=FCB 首址	AL=0 成功 AL=FFH 未找到
23H	取文件长度(结果在 FCB RR 中)	DS:DX=FCB 首址	AL=0 成功 AL=FFH 失败
17H	更改文件名(FCB 方式)	DS:DX=FCB 首址 (DS:DX+17)=新文件名	AL=0 成功 AL=FFH 失败
4EH	查找第一个匹配文件	DS:DX=文件路径名首址 CX=文件属性	若 CF=0 成功 DTA 中有该文件的信息 否则失败, AX=错误代码

功能号	功　　能	入口参数	出口参数
3. 目录操作功能			
4FH	查找下一个匹配文件	DTA 中有 4EH 得到的信息	(同 4EH)
43H	置/取文件属性	DS:DX=文件名首址 AL=0 取文件属性 AL=1 置文件属性(CX)	若 CF=0 成功 CX=文件属性(读时) 否则失败,AX=错误代码
57H	置/取文件日期和时间	BX=文件号 AL=0 取日期时间 AL=1 置日期时间(DX:CX)	若 CF=0 成功 DX:CX=日期和时间 否则失败,AX=错误代码
56H	更改文件号	DS:DX=老文件号首址 ES:DI=新文件号首址	
39H	建立一个子目录	DS:DX=目录路径串首址	若 CF=0 成功,否则失败
3AH	删除一个子目录	DS:DX=目录路径串首址	若 CF=0 成功,否则失败
3BH	改变当前目录	DS:DX=目录路径串首址	若 CF=0 成功,否则失败
47H	取当前目录路径名	DL=盘号 DS:SI=字符串首址	若 CF=0 成功 DS:SI=目录路径名首址 否则失败,AX=错误代码
4. 其他功能			
00H	程序结束，返回操作系统		
31H	终止程序并驻留在内存	AL=退出码 DX=程序长度	
4CH	终止当前程序,返回调用程序	AL=退出码	
4DH	取退出码		AL=退出码
33H	置取 Ctrl-Break 检查状态	AL=0 取状态 AL=1 置状态 (DL=0 关,DL=1 开)	DL=状态(AL=0 时)
25H	置中断向量	AL=中断类型号 DS:DX=中断服务程序入口	
35H	取中断向量	AL=中断类型号	ES:BX=中断服务程序入口
26H	建立一个程序段	DX=段号	
48H	分配内存空间	BX=申请内存数量 (以 16 字节为单位)	CF=0 成功 AX:0=分配内存首址 否则失败 BX=最大可用内存空间
49H	释放内存空间	ES:0=释放内存块的首址	CF=0 成功 否则失败,AX=错误代码

续表

功能号	功　　能	入口参数	出口参数
4. 其他功能			
4AH	修改已分配的内存空间	ES=已分配的内存段地址 BX=新申请的数量	CF=0 成功 AX:0=分配内存首址 否则失败 BX=最大可用内存空间
2AH	取日期		CX:DX=日期
2BH	置日期	CX:DX=日期	AL=0 成功 AL=FFH 失败
2CH	取时间		CX:DX=时间
2DH	置时间	CX:DX=时间	AL=0 成功 AL=FFH 失败
30H	取 DOS 版本号		AL=版本号, AH=发行号
38H	置/取国家信息	DS:DX=信息存放地址 AL=0	CF=0 成功 DS:DX=信息区地址

附录 E　BIOS 软中断简要列表

中　断	功 能 简 介
INT 10H	屏幕显示(共 16 个功能) 0 置显示模式 1 设置光标大小 2 置光标位置 3 读光标位置 5 置当前显示页 6 上滚当前页 7 下滚当前页 8 读当前光标位置处的字符及属性 9 写字符及属性到当前光标位置处 10 写字符到当前光标位置处 11 置彩色调色板 12 在屏幕上画一个点 13 读点 14 写字符到当前光标位置处,且光标前进一格 15 读当前显示状态 16 写字符串
INT 13H	磁盘输入输出(共 6 个功能) 0 磁盘复位 1 读磁盘状态 2 读指定扇区 3 写指定扇区 4 检查指定扇区 5 对指定磁道格式化
INT 14H	异步通信口输入输出(共 4 个功能) 0 初始化 1 发送字符 2 接收字符 3 读通信口状态
INT 16H	键盘输入(共 3 个功能) 0 读键盘 1 判别有无按键 2 读特殊键标志
INT 17H	打印机输出(共 3 个功能) 0 读状态 1 初始化 2 打印字符
INT 1AH	读写时钟参数(共 8 个功能) 0 读当前时钟 1 设置时钟 2 读实时钟 3 设置实时钟 4 读日期 5 设置日期 6 设置闹钟 7 复位闹钟

参 考 文 献

[1] Barry B. Brey. The Intel Microprocessors 8086/8088，80186/80188，80286，80386，80486，Pentium，Pentium Pro，and Pentium Ⅱ Processors Architecture，Programming，and Interfacing[M]. 7 版. 北京：机械工业出版社，2006.

[2] 黎明，等. 计算机硬件技术基础[M]. 北京：中国铁道出版社，2006.

[3] 张功萱，等. 计算机组成原理[M]. 北京：清华大学出版社，2005.

[4] 艾德才. 微型计算机(Pentium 系列)原理与接口技术[M]. 北京：高等教育出版社，2004.

[5] 周明德. 微型计算机系统原理及应用[M]. 4 版. 北京：清华大学出版社，2002.

[6] 郑学坚，朱定华. 微型计算机原理及应用[M]. 4 版. 北京：清华大学出版社，2013.

[7] 张菊鹏，等. 计算机硬件技术基础[M]. 北京：清华大学出版社，1997.

[8] Intel Corporation. The 8086 Family User's Manual. 1979.

[9] 吴宁，陈文革，等. 微型计算机原理与接口技术[M]. 西安：西安交通大学出版社，2009.